A polynomial-time algorithm for a class of protein threading problems

Ying Xu and Edward C. Uberbacher

Abstract

This paper presents an algorithm for constructing an optimal alignment between a three-dimensional protein structure template and an amino acid sequence. A protein structure template is given as a sequence of amino acid residue positions in three-dimensional space, along with an array of physical properties attached to each position; these residue positions are sequentially grouped into a series of core secondary structures (central helices and \(\beta\) sheets). In addition to match scores and gap penalties, as in a traditional sequence–sequence alignment problem, the quality of a structure–sequence alignment is also determined by interaction preferences among amino acids aligned with structure positions that are spatially close (we call these 'long-range interactions'). Although it is known that constructing such a structure–sequence alignment in the most general form is NP-hard, our algorithm runs in polynomial time when restricted to structures with a 'modest' number of long-range amino acid interactions. In this work, long-range interactions are limited to interactions between amino acids from different core secondary structures. Dividing the series of core secondary structures into two subseries creates a cut set of long-range interactions. If we use \(N\), \(M\) and \(C\) to represent the size of an amino acid sequence, the size of a structure template, and the maximum cut size of long-range interactions, respectively, the algorithm finds an optimal structure–sequence alignment in \(O(2^{C}NM)\) time, a polynomial function of \(N\) and \(M\) when \(C = O(\log(N + M))\). When running on structure–sequence alignment problems without long-range intersections, i.e. \(C = 0\), the algorithm achieves the same asymptotic computational complexity of the Smith–Waterman sequence–sequence alignment algorithm.

Introduction

Protein threading provides an important approach to predicting the three-dimensional (3-D) structure of a protein from its sequence, and has the potential to be considerably more computationally tractable than folding simulation using molecular dynamics (Brooks et al., 1988) or genetic algorithms (Dandekar and Argos, 1996). It concerns finding an optimal alignment between an amino acid sequence and a 3-D protein structure template (Godzik et al., 1992; Jones et al., 1992; Sippl and Weitckus, 1992; Bryant and Lawrence, 1993; Johnson et al., 1993; Ouzounis et al., 1993; Flockner et al., 1996; Lathrop and Smith, 1996; Madej et al., 1996). Given a database of known protein structures, such a structure–sequence alignment can be used to predict the structure of an anonymous amino acid sequence.

In the protein threading problem, a protein structure template is given as a sequence of amino acid residue positions in 3-D space, along with an array of physical properties attached to each position; these structure positions are sequentially grouped into a series of core secondary structures. In this study, only \(\alpha\)-helices and \(\beta\) sheets are considered, and loops are removed. For more discussion on this matter, we refer the reader to Lathrop and Smith, (1996). Aligning ('threading') an amino acid sequence with a structure template involves assigning amino acids of the sequence to positions in the structure so that: (i) each amino acid is assigned to at most one position; (ii) each position has at most one amino acid assigned to it; and (iii) the aligned elements keep their original relative order. A fitness function is used to measure the quality of each individual assignment of an amino acid to a structure position based on how well they match 'physically' (match between the amino acid and the array of physical properties). Unaligned amino acids and structure positions are penalized by a penalty function. Interactions may exist among amino acids assigned to structure positions that are spatially close, but not necessarily close in the sequence. Certain assignments of amino acids to a group of interacting structure positions may be preferred over others. The overall quality of a structure–sequence alignment is determined by the fitness of each individual assignment of an amino acid to a structure position, interactions between amino acids that are spatially close, and gaps existing in the alignment.

In formulating the protein threading problem, we follow the basic assumptions of Lathrop and Smith...
(1996). We assume that (i) when aligning a sequence with a structure template, successive positions of each core secondary structure are occupied by adjacent amino acids in the sequence, and alignment gaps are confined to the connecting loop regions (Lathrop and Smith, 1996) (this restriction is relaxed in the Summary); (ii) only pairwise interactions are considered and interaction preferences are additive; (iii) the fitness of individual assignments of amino acids to structure positions is additive; (iv) variable-length gaps are allowed and gap penalties are additive; and (v) terms from (ii), (iii) and (iv) are additive. Based on these assumptions, Lathrop (1994) has shown that the protein threading problem is NP-hard.

Two main strategies have been devised to tackle this computationally difficult problem: exhaustive search (Bryant and Lawrence, 1993) and implicit exhaustive search like branch-and-bound (Lathrop and Smith, 1996), and heuristic methods including Monte Carlo methods (Bryant and Altschul, 1995), Gibbs Sampling (Madej, et al., 1995), and iterative methods (Godzik et al., 1992). Heuristic algorithms like Monte Carlo methods, etc., are, in general, not guaranteed to find an optimal structure–sequence alignment within a finite amount of time, and will usually stop at some local optimal solution. Methods like branch-and-bound find an optimal structure–sequence alignment by implicitly searching through the whole space of possible alignments. They gain computational efficiency by excluding portions of the search space that it is not necessary to search, based on domain-specific knowledge. In the worst case, the whole search space may have to be searched explicitly.

We have developed a divide-and-conquer algorithm that guarantees to find an optimal structure–sequence alignment, and runs in polynomial time when restricted to protein structures that have a 'modest' number of pair interactions. The basic idea of the algorithm can be described as follows. It first divides both the sequence and the structure template into two subsequences and substructures in a number of different ways, and recursively solves the optimal structure–sequence alignment problem for each pair of subsequence and substructure; then it combines the optimal alignments for the subproblems to construct an optimal alignment for the original problem. The sequence and the structure are divided in such ways that an optimal alignment is guaranteed to be obtainable from the optimal alignments for the subproblems.

The bottleneck of the algorithm lies in dealing with pair interactions. Consider the given series of core secondary structures. Dividing the series between two consecutive core secondary structures creates a cut set of interactions, the ones interacting across the division. We call these long-range interactions. If we use C to denote the maximum cut size of long-range interactions, the algorithm runs in \( O(2^C NM) \) time, where \( N \) and \( M \) are the sequence size and structure size, respectively. Hence, the algorithm runs in polynomial time when restricted to structures satisfying \( C = O(\log(N + M)) \).

System and methods

One version of the protein threading algorithm presented in this paper has been implemented using C programming language on a Sparc 20 workstation under operating system SunOS 4.1.2.

Algorithm

This section gives a formal definition of a protein threading problem and presents an algorithm for solving the problem. Throughout this paper, \( \log() \) means \( \log_2() \).

Problem definition

Given an amino acid sequence \( s = s_1s_2 \ldots s_N \) and a 3-D protein structure template \( S = S_1S_2 \ldots S_M \), where each \( s_i \) represents an amino acid and each \( S_j \) represents a structure position along with its attached physical properties; \( S \) is grouped into a series of core secondary structures \( C = (C_1, \ldots, C_m) \). The goal is to align \( s \) with \( S \) while allowing gaps only in between core secondary structures so that the alignment quality is optimized.

Let \( \bar{s} = \bar{s}_1 \bar{s}_2 \ldots \bar{s}_p \) and \( \bar{S} = \bar{S}_1 \bar{S}_2 \ldots \bar{S}_p \) be an alignment (Smith and Waterman, 1982) of \( s \) and \( S \) with:

\[
\bar{s}_1 \quad \bar{s}_2 \quad \ldots \quad \bar{s}_{p-1} \quad \bar{s}_p \\
\bar{S}_1 \quad \bar{S}_2 \quad \ldots \quad \bar{S}_{p-1} \quad \bar{S}_p
\]

where \( \bar{s}_i \) is either an element of \( s \) or \( \phi \) (representing a gap), and similarly \( \bar{S}_i \) is an element of \( S \) or \( \phi \), and \( \max\{N, M\} \leq p \leq N + M \). In this section, we consider only the alignments that have gaps confined to the connecting loop regions, or formally that have \( s_{j+x} \) aligned with \( S_{i+x} \) if \( s_j \) is aligned with \( S_i \), and \( S_j \) and \( S_{i+x} \) belong to the same core secondary structure \( C_x \), for any integral \( x \) (Lathrop and Smith, 1996). We use \( \mathcal{A}(s, S) \) to denote the set of all possible such structure–sequence alignments between \( s \) and \( S \), and \( \mathcal{I}(S) \) to denote the set of long-range pair interactions between positions from different core secondary structures. (This set can be constructed from \( S \) based on its elements' geometric positions and user's specification. In this study, we assume that it is given.)

For each pair \( \bar{s}_i \) and \( \bar{S}_j \), \( F(\bar{s}_i, \bar{S}_j) \) denotes the alignment score of the two, which could be either a match score or a gap penalty depending on what \( \bar{s}_i \) and \( \bar{S}_j \) are ("F" for fitness). For each existing interaction between positions \( \bar{S}_i \) and \( \bar{S}_j \), the interaction preference of assigning \( \bar{s}_i \) and \( \bar{S}_j \)
to these two positions is given by \( P(\bar{s}_i, \bar{s}_j, \bar{S}_i, \bar{S}_j) \) ('P' for preference). The \( F() \) and \( P() \) values are given as two tables. The protein threading problem is defined as finding a structure–sequence alignment that minimizes the following function:

\[
\min_{(\bar{s}, \bar{S}) \in \mathcal{A}(s, S)} \left\{ \sum_{1 \leq i, j \leq \rho} F(\bar{s}_i, \bar{S}_i) + \sum_{(i,j) \in \mathcal{I}(S)} P(\bar{s}_i, \bar{s}_j, \bar{S}_i, \bar{S}_j) \right\}
\]  

(1)

Figure 1 illustrates the basic problem of protein threading.

In this formulation of the protein threading problem, we do not consider pair interactions within the same core secondary structure; also we do not allow gaps within a core secondary structure in the alignment. Gap penalties are assumed to be independent of each other. Most of these restrictions are placed for the simplicity of the presentation of the algorithm. Relaxation of these restrictions is addressed in the Summary.

**Algorithm description**

This subsection presents an algorithm for solving the optimization problem (1). We use \( s[i,j] \) to represent the subsequence of \( s \) from position \( i \) to position \( j \), and similarly for substructure \( S[i,j] \).

The algorithm first divides the structure \( S \) into two parts, \( S[1,k] \) and \( S[k+1,M] \), as shown in Figure 2a. The choice of \( k \) may affect the computational efficiency, while the correctness of the algorithm outlined below holds for any \( k \in [1, M - 1] \). Such a division creates a cut set of interactions, the ones interacting across the division (see Figure 2a). Let \( T \) represent the set of residue positions in \( S[1,k] \) having interactions with \( S[k+1,M] \) (labeled as circles in Figure 2a).

When dividing \( S \), we treat the left and right halves differently. (i) We replicate \( T \) and then attach it to \( S[k+1,M] \) through its interaction links (as shown in Figure 2b), to keep the pair interactions intact after division. (ii) We attach a separate replication of \( T \) to \( S[1,k] \) through a set of artificial interaction links (note that adding the artificial interactions will not change the cut size of \( S[1,k] \)); create an interaction between each attached element with its corresponding element in \( S[1,k] \); and this artificial interaction has a preference value \( P() = 0 \) when its two ends are assigned with a pair of the same amino acid or gap, otherwise \( P() = \infty \) (see Figure 2c). With such a definition of \( P() \) for an artificial interaction, the algorithm can ‘force’ positions in \( S[1,k] \) having outside interactions to be aligned with particular amino acids or gaps. The reason for introducing artificial interactions instead of directly putting constraints on the interacting positions in \( S[1,k] \) is to solve the alignment problems for the left and right halves uniformly.

We call the attached elements terminals. Terminals will not be a part of a structure–sequence alignment, rather they serve as a set of constraints. Each possible assignment of amino acids or a gap to the terminals changes the alignment quality by adding to the alignment score the preference values \( P() \) determined by this assignment and the alignment.

In the next step, the algorithm finds an optimal alignment between \( s[i,i] \) and \( S[1,k] \) with terminals \( T \), and also finds an optimal alignment between \( s[i+1,N] \) and \( S[k+1,M] \) with terminals \( T \), for each \( i \in [1, N - 1] \) and each possible combination of assignments to \( T \). Note that each of the 20 amino acids and the gap defines one assignment to a terminal. Hence, there are \( 21|T| \) combinations of possible assignments for a set of \( |T| \) terminals.
The algorithm finds optimal alignments for the subproblems recursively using these two steps.

We now give the pseudo code of the structure-sequence alignment algorithm, whose correctness is implied by equation (2) in the next subsection. The inputs to the algorithm are (i) the starting and ending positions, \( k_1 \) and \( k_2 \), of an amino acid subsequence, (ii) the starting and ending positions, \( m_1 \) and \( m_2 \), of a protein substructure, and (iii) a list of terminals \( T \) attached to the current subsequence and a list \( A \) of amino acids or gaps assigned to \( T \). The output is the score of an optimal alignment between the given (sub)sequence and (sub)structure under the constraint that the attached terminals \( T \) are assigned with a particular set of amino acids \( A \).

**Procedure** threading\((k_1, k_2, m_1, m_2, T, A, \text{score})\)

1. **begin**
2. let \( m' \) be the index of the last position of the core secondary structure starting at \( m_1 \);
3. divide the terminals \( T \) into \( T_1 \) and \( T_2 \) with \( T_1 \) containing interactions with \( S[m_1, m'] \) and \( T_2 \) containing interactions with \( S[m'+1, m_2] \), and divide \( A \) into \( A_1 \) and \( A_2 \) accordingly;
4. calculate the interactions between \( S[m_1, m'] \) and \( S[m'+1, m_2] \), and the corresponding terminals \( T_0 \);
5. \( \text{score} \leftarrow \infty \);
6. for each combination of assignments \( A_0 \) to \( T_0 \) and each \( k \in [k_1 - 1, k_2 - 1] \) do
7. **begin**
8. call align\((k_1, k, m_1, m', T_0 \cup T_1, A_0 \cup A_1, \text{score}_1)\);
9. if \( (m' < m_2) \) then call threading\((k+1, k_2, m'+1, m_2, T_0 \cup T_2, A_0 \cup A_2, \text{score}_2)\);
else \( \text{score} \leftarrow 0 \);
10. if \( \text{score} > \text{score}_1 + \text{score}_2 \) then \( \text{score} \leftarrow \text{score}_1 + \text{score}_2 \);
11. **end**
12. **end**

align() is a subprocedure for aligning a sequence with a core secondary structure. The difference between this alignment and a general structure-sequence alignment is that no gaps are allowed within a core secondary structure (although we do allow a whole core secondary structure to be aligned with gaps). The following procedure finds a structure-sequence alignment under the constraint of a fixed set of terminal assignments. It works even when a core secondary structure has internal interactions.

**Procedure** align\((k_1, k_2, m_1, m_2, T, A, \text{score})\)

1. **begin**
2. \( \text{score} \leftarrow \sum_{i \in [k_1, k_2]} F(s_i, \phi) + \sum_{i \in [m_1, m_2]} F(\phi, s_i) \);
/* set the initial alignment to gaps */
3. for each \( k \in [k_1, k_2 - (m_2 - m_1)] \) do
4. **begin**
5. \( \text{score}_{\text{temp}} \leftarrow \sum_{i \in [k, k+(m_2-m_1)]} F(s_i, S_1) + \sum_{(i,j) \in T} P(a(i), a(j), S_1, S_2) + \sum_{i \in [k, k-1]} F(s_i, \phi) + \sum_{i \in [k+(m_2-m_1)+1, k_2]} F(s_i, \phi) \); /*
6. \( T \) : the set of interacted positions determined by \( T \), and
7. \( a(x) : A(x) \) if \( x \in T \) otherwise \( s_x \).
8. */ (add to \( \text{score}_{\text{temp}} \) the internal interaction preference values if they exist.)
9. if \( \text{score} > \text{score}_{\text{temp}} \) then \( \text{score} \leftarrow \text{score}_{\text{temp}} \);
10. **end**
11. **end**

A careful reader may have noticed that **Procedure** threading() can be implemented much more efficiently since, in its current form, a large amount of recalculation has been done. Note that on line 9, threading\((k+1, k_2, m'+1, m_2, T_0 \cup T_2, A_0 \cup A_2, \text{score}_2)\) will be called not only by threading\((k_1, k_2, m_1, m_2, T, A, \text{score})\), but also by, for example, threading\((k_1 - 1, k_2, m_1, m_2, T, A, \text{score})\), or in general by any subsequence \( s[a, b] \) and any substructure \( S[c, d] \) with \( [k_1, k_2] \subseteq [a, b] \) and \( [m_1, m_2] \subseteq [c, d] \). The same can also be said about **Procedure** align() on line 8. To avoid recalculation, we replace lines 8 and 9 of **Procedure** threading() by the following.

8'. If align\((k_1, k, m_1, m', T_0 \cup T_1, A_0 \cup A_1, \text{score}_1)\) has been called
then \( \text{score} \leftarrow \) the score from the first call;
else call align\((k_1, k, m_1, m', T_0 \cup T_1, A_0 \cup A_1, \text{score}_1)\),
and record the score;
9'. If threading\((k+1, k_2, m'+1, m_2, T_0 \cup T_2, A_0 \cup A_2, \text{score}_2)\)
has been called
then \( \text{score} \leftarrow \) the score from the first call;
else begin
if \( (m' < m_2) \) then call threading\((k+1, k_2, m'+1, m_2, T_0 \cup T_2, A_0 \cup A_2, \text{score}_2)\);
else \( \text{score} \leftarrow 0 \);
record the score;
end

To calculate the score of an optimal structure-sequence alignment between a sequence \( s \) and a structure \( S \), we need to call threading\((1, |s|, 1, |S|, \emptyset, \emptyset, \text{score})\). To recover an optimal alignment that achieves the optimal scoring, some simple bookkeeping needs to be done to record branches that achieve the optimal scoring on each level of the recursion. This can be done without changing the asymptotic complexity of the algorithm.

**Proof of correctness**

This subsection shows the correctness of the algorithm by
Fig. 3. Implementation of Procedure threading(). The short rectangles in both (a) and (b) represent one secondary structure, and the long rectangles represent a series of secondary structures. The curved lines represent an optimal structure–sequence alignment. Dotted arrows indicate the matched portions in an optimal structure–sequence alignment.

Showing that there is sufficient information to construct an optimal alignment between s and S from the optimal alignments between the subsequences and substructures as described in the previous subsection. We give an informal proof [a formal proof can be obtained by following the same argument and using the definition of the objective function (1)] to the statement that there is an \( i^* \in [1, N - 1] \) and a list of assignments \( A^* = (a_1, a_2, \ldots, a_t) \) to S so that an optimal alignment between s and S can be constructed by merging the optimal alignment between \( s[i, i^*] \) and \( S[1, k] \) with terminal assignments \( A^* \), and the optimal alignment between \( s[i + 1, N] \) and \( S[k + 1, M] \) with terminal assignments \( A^* \), for any \( k \in [1, M - 1] \), where \( T \) is the set of terminals formed when dividing \( S[1, k] \) and \( S[k + 1, M] \).

Let \( A_0 \) be an optimal alignment between s and S, with \( s_j \) assigned to \( S_k \), and \( s_i \) assigned to \( S_{i + 1} \) in \( A_0 \). We use \( A \) to denote the list of amino acids assigned to the structure positions in \( S[1, k] \) having interactions in \( S[k + 1, M] \). By the optimality of \( A_0 \), we know that for any \( q \in [i, j] \), the portion of \( A_0 \) between \( s[i, q] \) and \( S[1, k] \) forms an optimal alignment between \( s[i, q] \) and \( S[1, k] \) under the constraint that the structure positions of \( S[1, k] \) having interactions with \( S[k + 1, M] \) are aligned with \( A \). This is equivalent to saying that the portion of \( A_0 \) between \( s[i, q] \) and \( S[1, k] \) forms an optimal alignment between \( s[i, q] \) and \( S[1, k] \) under the constraint that \( S[1, k] \)'s terminals are assigned with \( A \). Similarly, the portion of \( A_0 \) between \( s[q + 1, N] \) and \( S[k + 1, M] \) forms an optimal alignment between \( s[q + 1] \) and \( S[k + 1, M] \) under the constraint that \( S[k + 1, M] \)'s terminals are assigned with \( A \). This implies that if we choose \( i^* = q \) and \( A^* = A \), we have proved the above statement.

If we use \( \text{score}(s, S) \) to denote the score of an optimal alignment between s and S [the one minimizing the objective function (1)], and \( \text{score}(T, A)(s[i, i], S[1, k]) \) and \( \text{score}(T, A)(s[i + 1, N], S[k + 1, M]) \) the scores of optimal alignments between \( s[i, i] \) and \( S[1, k] \), and \( s[i + 1, N] \) and \( S[k + 1, M] \), respectively, under the constraint that their terminals \( T \) are assigned with amino acids or gaps \( A \), the above proof implies that for any fixed \( k \in [1, M - 1] \):

\[
\text{score}(s, S) = \min_{i \in [1, N - 1], A} \{ \text{score}(T, A)(s[i, i], S[1, k]) + \text{score}(T, A)(s[i + 1, N], S[k + 1, M]) \}
\]

Complexity analysis

We now give more implementation details of Procedure threading() and Procedure align(), and analyze their computational complexity.

To avoid recomputation in the implementation of the algorithm, we keep a table to record alignment scores of optimal alignments between substructure \( S[k, M] \) and subsequence \( s[i, N] \), for the starting position \( k \) of each core secondary structure, each \( i \in [1, N] \), and each possible combination of assignments to the terminals of \( S[k, M] \). Entries of the table are filled in the increasing order of i and k. To store the table takes \( O(2^{1+C}NM) \) space, and hence the total space used is \( O(2^{1+C}NM) \) since the size of this table dominates the total space used in the algorithm.

Consider a substructure \( S[k, M] \) and a subsequence \( s[i, N] \) (as defined above). Let \( S[k, k' - 1] \) be the starting core secondary structure of \( S[k, M] \) (see Figure 3). We have the following observation on how an optimal alignment between \( S[k, M] \) and \( s[i, N] \) is related to an optimal alignment between \( S[k, M] \) and \( s[i - 1, N] \). Let \( A_0 \) denote an optimal alignment between \( S[k, M] \) and \( s[i, N] \) (see Figure 3a), and \( A_1 \) denote an optimal alignment between \( S[k, M] \) and \( s[i - 1, N] \) under the constraint that \( S[k, k'] \) is aligned with \( s[i - 1, i + k' - k - 1] \) without gaps (see Figure 3b). We can show that either \( A_1 \) or extending \( A_0 \) by including alignment pair \( (s_{i - 1}, \phi) \) gives an optimal alignment between \( S[k, M] \) and \( s[i - 1, N] \).

This implies that to compute an optimal alignment between \( S[k, M] \) and \( s[i - 1, N] \), when having an optimal alignment between \( S[k, k'] \) and \( s[i - 1, N] \), we only need to compute an optimal alignment between \( S[k, M] \) and \( s[i - 1, i + k' - k - 1] \) under the constraint that \( S[k,k'] \) is aligned with \( s[i - 1, i + k' - k - 1] \) without gaps. This optimal alignment can be computed as follows: let \( T \) denote the terminals formed when dividing \( S[k, k' - 1] \) and \( S[k, k'] \), for each possible combination of assignments to \( T \) compute the alignment score between \( S[k, k' - 1] \) and \( s[i - 1, i + k' - k - 1] \), and get the optimal alignment score between \( S[k', M] \) and \( s[i + k' - k, N] \) by a table...
look-up operation (note that this score has been computed already), and then sum up these two scores; the lowest score among all possible assignments is compared to the alignment score of the extended $A_0$ (see the above paragraph), and the lower one of the two is recorded as the alignment score between $S[k, M]$ and $s[i, N]$.

It can be checked that the above calculation takes $O(2^{1^C(k' - k)})$ time, for each $i$ and each such $k$. Hence, the total time, for all possible $i$ and $k$, is $O(2^{1^C NM})$. Also note that this part of the algorithm dominates the computation time of the whole algorithm, and hence the time complexity of the algorithm is $O(2^{1^C NM})$.

Summary

Research is currently under way to test the effectiveness of the algorithm on known protein structures. Issues we are evaluating include (i) how general our algorithm is under the assumption of a 'modest' number of long-range interactions, and (ii) how efficient our algorithm is in real CPU time.

Generality

In addition to the protein structure size $M$ and the amino acid sequence size $N$, we have introduced a new (independent) parameter $C$, the maximum cut size, in our protein threading algorithm. $C$ measures the intrinsic 'density' of the long-range pair interactions in a folded protein structure. Our algorithm achieves a polynomial computation time when restricted to protein structures satisfying $C = O(\log(N + M))$. Alternatively, $C$ can be considered as a parameter of the algorithm, which controls the level of approximation to the actual long-range pair interactions in a folded protein structure. For protein structures with 'dense' pair interactions, we have applied the following strategy: extract a set of the most significant long-range pair interactions which satisfy our assumption of having a 'modest' number of interactions; and treat the rest as 'weaker' interactions using methods like 'frozen approximation' (Godzik et al., 1992). We have developed an algorithm (Y. Xu and E.C. Uberbacher, unpublished results) to extract the subset of the most significant long-range pair interactions using a generalized bipartite graph-matching algorithm. Research is currently under way to study the effectiveness of this strategy.

Efficiency

Note that when dividing a structure into two substructures, each combination of assignments to the terminals should correspond to a (sub)structure-(sub)sequence alignment in the other half of the division. Such a combination of assignments should be omitted if no corresponding alignment exists. For example, consider terminals $a_1, a_2, \ldots, a_C$ and assume that they are from the same secondary structure. For the simplicity of discussion, we further assume that $a_1, a_2, \ldots, a_C$ are consecutive elements in the secondary structure. Because of our assumption that no gaps are allowed within a secondary structure alignment, only $O(\min\{2^C, N\})$ possible combinations of assignments need to be considered. More generally, let $T = \{a_1, \ldots, a_C\}$ denote a set of terminals at a division point, and $T$'s elements be from $k$ different secondary structures. By an analogy to the above argument, there are $O(\min\{2^C, N^A\})$ possible combinations of assignments to $T$. Hence, by taking advantage of this observation in our implementation, the running time of our algorithm reduces to $O(\min\{2^C, N^A\} NM)$, where $k$ is the maximum number of secondary structures that the terminal elements are from over all division points. This is a polynomial function of $M$ and $N$ when either $C = O(\log(M + N))$ or $k$ is a constant.

Theoretically, the algorithm remains a polynomial-time algorithm as long as $C$ grows no faster than the logarithm of the size of the 3-D protein structure template and the amino acid sequence, or $k$ stays as a constant. Our preliminary results have shown that the algorithm finds an optimal structure–sequence alignment within a reasonable amount of time (up to a few CPU minutes) with the maximum cut size $C \leq 5$, or $k \leq 3$ for protein structures and amino acid sequences up to ~400 bases. Based on our current understanding of the problem, we expect to be able to improve the maximum cut size $C$ further to close to 10 and $k$ up to 5, and keep the computational time practical. On a test set of 147 proteins with sizes ranging from 100 to 800 amino acids, we calculated energies for long-range side chain–side chain interactions. Interactions for a distance $>10\AA$ between the centroids of the side chains are ignored. When the top 10% of the most significant interactions are considered, ~79% (116/147) of the proteins have $C < 10$ and 86% (126/147) have $k \leq 5$.

One simple way to make the algorithm run faster is to group the 20 amino acids into a small number of groups containing ‘similar’ amino acids, say four, at the expense of losing some threading accuracy. Then the algorithm would run in $O(\min\{5^C, N^A\} NM)$ time.

Extensions

(i) A simple modification allows our algorithm to use an affine function of the form:

$$p + \sum_i F(s_i, \phi) + \sum_i F(\phi, S_i)$$

to penalize a maximal run of gaps. This modification does not change the asymptotic running time of the algorithm.
(ii) With some minor modification to Procedure $\text{align}()$, our algorithm allows core elements to be aligned with gaps at the two ends of a core secondary structure, which turns out to be necessary when aligning core elements of very different sizes.

(iii) One way to allow gaps within a core secondary structure is to replace our current $\text{align}()$ procedure by a Smith-Waterman type of algorithm.

In summary, we have developed a polynomial-time algorithm for the protein threading problem on structures with 'modest' long-range pair interactions. With a guaranteed optimal alignment for any given set of alignment fitness function, gap penalty function and long-range pair interaction function, and a guaranteed polynomial running time, the algorithm should provide molecular biologists with a powerful tool in evaluating the effectiveness of different scoring functions for protein threading problems.

Acknowledgements

This research was supported by the United States Department of Energy, under contract DE-AC05-84OR21400 with Lockheed Martin Energy Systems, Inc. The authors would like to thank the anonymous reviewers for their critical comments and suggestions, which have helped improve the presentation of this paper. The authors also want to express their gratitude to Dr. J. Ralph Einstein for calculating the long-range interactions and tabulating the data.

References


