Chapter 12, pp. 227–283, in
"Computational Methods in Molecular Biology"
editors Steven Salzberg, David Searls, and Simon Kasif.
Elsevier Press, Amsterdam, 1998

Analysis and Algorithms for Protein Sequence-Structure Alignment

by

Richard H. Lathrop(1), Robert G. Rogers Jr.(2), Jadwiga Bienkowska(2),
Barbara K. M. Bryant(3), Ljubomir J. Buturović(4), Chrysanthe Gaitatzes(2),
Raman Nambudripad(5), James V. White(2,6), Temple F. Smith(2)

(1) Department of Information and Computer Science
444 Computer Science Building
University of California, Irvine
Irvine, CA 92697-3425

(2) BioMolecular Engineering Research Center
Boston University
36 Cummington Street
Boston, MA 02215

(3) Millennium Pharmaceuticals, Inc.
640 Memorial Drive
Cambridge, MA 02139

(4) Incyte Pharmaceuticals, Inc.
3174 Porter Drive
Palo Alto, CA 94304

(5) Molecular Computing Facility
Beth Israel Hospital
330 Brookline Avenue
Boston, MA 02215

(6) TASC, Inc.
55 Walkers Brook Drive
Reading, MA 01867

Chapter Overview

This chapter discusses analytic and algorithmic results for computational protein structure prediction by protein sequence-structure alignment, an approach also known as protein threading. Biological results are beyond the scope of this chapter, but may be found in [1, 2, 3, 4]. See also the chapter by David Jones in this volume, which discusses another approach to protein threading.

The chapter visits in turn: motivation, intuition, formalization, analysis, complexity, algorithm, computational cost, discussion, and conclusions. The early sections are tutorial in nature; the rest represent original research results. The overall conclusions are that: (1) computational techniques can render vast search spaces approachable by exploiting natural constraints; and (2) advances in knowledge-based objective functions and protein structural environment definitions represent an important opportunity for future progress.

A long-range goal of this work is to integrate structural and functional pattern recognition. The reader will notice that gapped block alignment is conceptually similar to block patterns, consensus patterns, weight matrices, profile patterns, and hierarchical patterns, among many other gapped block pattern methods (reviewed in [5]). Combined structural and functional pattern recognition is likely to prove more powerful than either one alone.

1 Introduction

Simply stated, the protein folding problem is to transform information. The input is a string of characters drawn from an alphabet of 20 letters. In the simplest case, the desired output annotates each character with three numbers, giving its XYZ coordinates in the protein's three-dimensional folded shape. Surprisingly, in many cases these coordinates are unique and depend only on the input string. There, protein structure prediction from sequence simply transforms implicit information into an explicit final form.

The protein folding problem is also the premiere computational problem confronting molecular biology today: it has been called the "holy grail of molecular biology" and "the second half of the genetic code" [6]. It is important because the biological function of proteins (enzymes) underlies all life, their function is determined by their three-dimensional shape, and their shape is determined by their one-dimensional sequence. The importance of computational solutions is escalating rapidly due to the explosion of sequences and genomes becoming available, compared to the slow growth in the number of experimentally determined three-dimensional protein structures.

The problem is unusually accessible to computer scientists because it is (in its essence) a pure information processing transformation, from implicit to explicit. No single computer program would so transform the face of experimental molecular biology practice today as one that correctly, reliably, and rapidly computed this function. It is a Grand Challenge problem for computer science [7].

1.1 Why is it hard?

The problem, although simply stated, is quite difficult. The process by which nature folds the string is complicated, poorly understood, and most likely the global sum of a large number of weak, local, interacting effects. Quantum mechanics provides a solution in principle, but the computation becomes intractable when confronted with the many thousands of atoms comprising a protein.

The direct approach to protein folding, based on modeled atomic force fields [8, 9] and approximations from classical mechanics, seeks to find the folded conformation having minimum free energy. This is difficult because a folded protein results from the delicate energetic balance of powerful atomic forces, and because the vast number of possible conformations poses a formidable computational barrier. The forces involved are often difficult to model accurately, and include stabilizing and destabilizing terms making large contributions of opposite sign summed over a very large number of atoms [10]. Thus, small cumulative approximation errors may dominate the smaller net stabilization. For technical reasons it is difficult to model surrounding water properly [9, 11], yet hydrophobic collapse is believed to be the main effect driving protein folding. Classical macroscopic parameters such as the dielectric constant become problematic at the atomic level. We may not know the protein's cellular folding context, which may include chaperone proteins, post-translational modifications, and hydrophobic interfaces to which the protein conforms. The search space may exceed 10^{50} plausible folded conformations even for medium-size proteins. Simulation time-steps are measured in femtoseconds while folding time scales of interest are measured in milliseconds, a ratio of 10^{12}. Unless sophisticated methods are used, the basic time-step computation is $\mathcal{O}(N^2)$ where N may approach 10^6 atoms with surrounding water. The simulation time may exceed 10^{12} CPU-years at current supercomputer speeds. The direct approach has been applied successfully to smaller molecules, but as yet faces stiff challenges for large proteins [11, 12], though recent versions using cruder force fields are promising [13, 14].

One important alternative approach is to use the wealth of information contained in already-known protein structures. The structures can serve as spatial folding templates, impose constraints on possible folds, and provide geometrical and chemical information. This is an attractive strategy because proteins exhibit recurring patterns of organization; there are estimated to be only around 1,000 to 10,000 different protein structural families [15, 16]. In this approach, the known structure establishes a set of possible amino acid positions in three-dimensional space. These template spatial positions generally include only the backbone atoms, though sometimes the implied beta carbon is used as well. The highly variable surface loops are not included in the template positions. Based on topological and physicochemical criteria, an alignment of an amino acid sequence to the set of positions in one such core template is chosen. Each amino acid of the sequence is given the three-dimensional coordinates of the template position to which it is aligned. Estimation of the complete structure still requires some means of assigning positions to the amino acids in the loop regions [17, 18], of assigning amino acid side chain orientations and packing [19, 20], and of searching the immediate structural neighborhood for a free energy minimum [8, 9]. In this chapter we focus on the choice of core template and the method of identifying the optimal alignment of the sequence to that core template.

Initially, such methods employed primary sequence string similarity between the candidate sequence and the structure's native sequence in order to perform the alignment ("homology modeling" or "homological extension"). Computing the sequence similarity yields a direct alignment of amino acids in the candidate's and structure's sequences [17, 21]. In cases where the sequence similarity is high this is still the most successful protein structure prediction method known. Unfortunately, it is of limited generality because novel sequences rarely have sufficiently high primary sequence similarity to another whose structure is known. Indeed, of the genomic sequences known at present, fully 40 percent have no homologs to any sequence of known function, let alone known structure [22].

1.2 Why threading?

Many evolutionarily unrelated sequences (non-homologs) contain similar domain folds or structural cores [15, 16, 23, 24, 25]. Recently, approaches have been devised which exploit this fact by aligning a sequence directly to a structure or structural model. The process of aligning a sequence to a structure and thereby guiding the spatial placement of sequence amino acids is referred to as "threading" the sequence into the structure [26, 27], and "a threading" means a specific alignment between sequence and structure (chosen from the large number of possible alignments). In this way "threading" specializes the more general term "alignment" to refer specifically to a structure (considered as a template) and a sequence (considered as being arranged on the template).

These new approaches exploit the fact that amino acid types have different preferences for occupying different structural environments (for example, preferences for being in alpha-helices or beta-sheets, or for being more or less buried in the protein interior). Additionally, some of the new approaches also exploit the fact that there appear to be distinct preferences for side-chain contact (e.g., contact potentials [28]), or more generally for spatial proximity (e.g., potentials of mean force [29]), as a function of those environments. For example, a buried charged residue may be more likely to be adjacent to another buried residue of opposite charge. These interaction preferences have been quantified statistically and used to produce a score function reflecting the extent to which amino acids from the sequence are located in preferred environments and adjacent to preferred neighbors. The known protein structures can be represented in a way that makes explicit the structural environment at each position, as well as the spatially adjacent structural positions. This done, the sequence can be aligned or threaded into the structure by searching for an alignment or threading that optimizes the score function. The optimal threading(s) maximize(s) the degree to which environment and adjacency preferences are satisfied. This has been a very active area of recent research, in part because it has been somewhat successful, and numerous scoring schemes and threading algorithms have been proposed. For reviews see [30, 31, 32, 33, 34, 35, 36, 37, 38], while for cautionary notes see [1, 4, 12, 39, 40, 41, 42].

2 Protein Threading — Motivating Intuitions

The logic behind the threading approach to the prediction of an amino acid sequence's expected three-dimensional fold is almost seductively simple, if not obvious. Given the extreme difficulty of any direct, de novo, or quantum level approach to protein structure prediction, combined with our esthetic sense as expressed in Occam's principle of parsimony [43], this seduction is understandable. In practice, however, as of this writing the threading approach has not yet lived up to our expectations [36]. The threading research challenge, only partially met at present, is to devise a theory of protein structure sufficiently information-rich to allow accurate prediction yet sufficiently concise to retain the simplicity of discrete alignment.

2.1 Basic ideas

Threading rests on two basic ideas: first, that there is a limited and rather small number of basic "protein domain core" folds or architectures found in nature; and, second, that some average over an entire sequence of amino acid propensities to prefer particular structural/solvent environments is a sufficient indicator for the recognition of native-like versus non-native-like folds. The first of these ideas is supported by our understanding of polymer chemistry, which suggests that there is a limited number of ways to fold a repetitive polymer of two basic unit types (amino acids are to a first approximation either hydrophilic or hydrophobic) in an aqueous environment. These are helical structures in which the helical repeat is synchronous with the polymer unit repeat, and the extended sheet structures in which neighboring polymer strands place the adjacent polymer repeat in synchrony. This results in six basic protein fold building blocks: helices and sheets, either of which can be all hydrophobic, all hydrophilic, or amphipathic (or two-sided). Now, given the requirement that a protein's interior is hydrophobic and its surface is hydrophilic, there appears to be a limited number of ways to pack these six building blocks together. Indeed the vast majority of currently determined structures fall into only a few core architectures, or arrangements of helices and sheets [16], as perhaps first clearly stated by Jane Richardson in her "taxonomy of proteins" [25]. Nonetheless, a major limitation of the threading approach is that if an appropriate core is not already present in the structure library, correct prediction is obviously impossible. Currently most of the estimated thousands of fold families remain unseen, and entirely novel folds appear frequently. Attempts to assemble structure fragments into a novel core include [44], [45], and [46].

The second threading concept, that the various preferences of the different amino acids for different structural environments can provide sufficient information to choose among alternate basic fold architectures, is less obvious. Protein structures, and even their functions, are known to be very robust to most single amino acid substitutions. This is a requirement of an evolving system. As has been noted many times, the thousands of distinct hemoglobin amino acid sequences all fold to nearly identical three-dimensional structures [47]. In addition, protein structures must be stable enough to tolerate some amino acids in very unfavorable positions, such as hydrophobic ones on the surface in a site designed to bind a hydrophobic substrate. It thus seems very unlikely that particular atomic details of particular amino acids determine the overall architecture of the fold. Given that many different amino acid pairs can produce nearly equivalent interactions, in terms of contact energies and packing density, it generally has been assumed that some average over these pairwise contact interactions determines relative positioning of the helices and sheets with respect to one another.

2.2 Common assumptions of current threading work

All current threading proposals replace the three-dimensional coordinates of the known structure by an abstract template description in terms of core elements and segments, neighbor relationships, distances, environments, and the like. This avoids the computational cost of atomic-detail molecular mechanics or dynamics [8, 9] in favor of a much less detailed, hence much faster, discrete alignment between sequence and structure. However, important aspects of protein structure, such as intercalated side-chain packing, excluded volume, constraints linking different environments, higher-order interactions, and so forth, also may be abstracted away. This depends on the theory employed.

The common assumptions underlying all threading proposals are as follows:

1. the known structures provide sets of template positions in three-dimensional space;

2. a sequence-structure alignment, or threading, specifies the placement of sequence residues into one of the sets of template positions;

3. different candidate threadings arise from different structural templates, different sequences, and different possible alignments of template positions and sequence amino acids; and

4. a score function (often statistical) can distinguish good from bad threadings.

2.3 Principal requirements for structure prediction

To predict accurately the structure of a novel protein sequence using the threading approach, it is necessary both to select the proper core template from a library of known examples ("fold recognition," figure 1), and to align the sequence to it correctly ("sequence-structure alignment," figure 2), simultaneously (figure 3). There are four components to any practical application of the threading approach to the prediction of the three-dimensional fold for an amino acid sequence:

(i) Construction of as complete as possible a library of potential core folds or structural templates.

(ii) An objective function (i.e., a score function) to evaluate any particular placement or alignment of a sequence into each core template.

(iii) Some means of searching over the vast space of possible alignments between a given sequence and each core template in the library for those that give the best total score.

(iv) A means of choosing the best template from among the best scoring alignments of a sequence to all possible templates.

Figures 1, 2, 3 about here.

This chapter focuses on the theory of selecting the best core templates and alignments (requirements iii and iv). It analyzes closely the consequences of choosing "best = globally highest conditional probability." For this case, it provides a probabilistic Bayesian theory that unifies core recognition and sequence-structure alignment (requirements iii and iv). The theory is Bayes-optimal because it rigorously selects cores and alignments that are globally most probable, regardless of the particular theory of protein structure adopted.

In contrast, the theory of protein structure does in fact determine the particular forms of the core templates and the score or objective function (requirements i and ii). We assume that the objective function may be interpreted as encoding the probability of observing a given sequence in a given alignment to a given core structure, and otherwise consider requirements (i) and (ii) to be arbitrary and fixed in advance. For approaches to requirements (i) and (ii) see references [27, 28, 37, 40, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57]. A rigorous probabilistic derivation of the objective function from Markov Random Field (MRF) theory may be found in White *et al.* [58] and Stultz *et al.* [59].

2.3.1 Library members (requirement i)

Library members (requirement i) variously are termed structures, folds, cores, folding motifs, folding patterns, topology fingerprints, three-dimensional profiles, contact profiles, adjacency graphs, spatial patterns, domains, structural models, tertiary templates, structural templates, core templates, and so on. Here we refer to them as core templates to emphasize the loop region variability; but the name has no special meaning and the analysis applies to many structure representations currently in the literature. Library members usually consist of abstractions of known structures, constructed by erasing atomic detail of specific native side-chains and variable or loop regions while retaining core backbone structural features. The core template is annotated with environmental features that describe local structural neighborhoods, such as spatial adjacencies, distances, angles, secondary structure, solvent accessibility or exposure, backbone parameters, and so on. The environmental features chosen depend upon the particular theory of protein structure adopted. Sometimes idealized coordinates replace database coordinates, allowing variable-length segments.

The core template corresponds to an annotated backbone trace of the secondary structure segments in the conserved core fold. Core segments are connected by variable loop or coil regions. Loops are not considered as part of the conserved fold, and are modeled by an arbitrary loop score function that is "sequence-specific," because it may depend upon the specific sequence residues forming the loop. In contrast, the gap penalty used in dynamic programming usually is a function of gap length only [21], and usually does not depend on the identity of sequence residues in the gap. Core template spatial "positions," implying a specific three-dimensional location, are abstracted to become spatially neutral core "elements," implying only a discrete place-holder that may be occupied by a residue from the sequence. Depending upon the requirements of the particular theory of protein structure adopted, the core template may record local structural environments, spatial neighbors, degree of solvent exposure, distances between core elements, and so on. In this way, the annotated core template organizes its core elements: each is embedded in an implied structural environment and interacts with structurally implied neighbors.

Pairs of elements are "neighbors" if they interact within the given score function. It is necessary to record on the core template whatever information the score function will use to assign scores to candidate threadings (e.g., Bryant & Lawrence [27] and Sippl [60] both record discretized distances). Such information comprises the abstract "structural environment" as seen by the threaded residues. If the score function quantifies individual residue preferences (e.g., for being more or less buried, or for certain secondary structures), the (singleton) structural environment at each element must be recorded. This is easily done by labeling the element. If the score function quantifies neighbor preferences (e.g., for being a certain distance apart), the (pairwise) structural environment between neighbors must be recorded. Several equivalent data structures have been used; the common theme is that certain pairs of elements are distinguished as neighbors, and the pair is labeled in some manner. One common data structure constructs a matrix having one row and one column for each element. Each cell contains the corresponding pairwise environment (e.g., a distance matrix [60] results when all elements are neighbors and the pairwise environment is Euclidean distance). An equivalent approach, the adjacency graph used here, constructs a graph with one vertex for each element. Neighbor elements are connected by a (directed) edge, and the edge is labeled with the pairwise environment. The edge in the graph corresponds to the cell in the matrix, and the edge label corresponds to the label contained in the cell. Related representations include adjacency matrix, contact graph, and so on. For pairwise interactions this is fully general, because each label could (in principle) specify a different 20×20 table with an arbitrary score for any possible pair.

2.3.2 Objective function (requirement ii)

Each distinct threading is assigned a score by a specified objective function or score function (requirement ii). This chapter generally restricts attention to score functions that can be computed by considering no more than two core segments at a time.

The objective function usually describes the degree of sequence-structure compatibility between sequence amino acid residues and their corresponding positions in the core template as indicated by the alignment (e.g., contact potentials, knowledge-based potentials, potentials of mean force, etc.). In the general case, the objective function may reflect the sequence residue types placed elsewhere in the structure. For example, a polar residue in a buried structural environment may be more likely to interact with a complementary polar residue in a neighboring buried position than with a hydrophobic residue there. In contrast, what we refer to below as the "singleton-only" objective function ignores interactions between pairs (or higher) of sequence residues. It is restricted to reflect only individual sequence residue preferences for the structural environments that annotate their core template position. For example, such a function may utilize the fact that a hydrophobic sequence residue may be more likely to occur in a buried structural environment than in an exposed one.

The general form of most scoring schemes proposed thus far consists of pseudo energies associated with each amino acid type for its placement in any given structural environment and neighbor relationships. Such pseudo energies are statistics derived from the observed occurrence frequencies for amino acid types in each structural environment among a set of determined structures. Frequencies are converted to a pseudo energy using Gibbs' or Boltzmann's relationship between system state energy and state probability, or simply by taking negative logarithms of occurrence frequency-derived probabilities or odds ratios. The structural environments used have included the degree of solvent exposure, the type of secondary structure, the "distance" between spatial neighboring amino acids, and so on, across the types of amino acid residues that are observed as typical neighbors.

2.3.3 Alignment (requirement iii)

The alignment of the sequence to a given core template (requirement iii) usually is selected by searching for the best alignment under the objective function. An alignment is optimal if its score is the global minimum score, and near-optimal if it approximates this. A search method is said to be exact if it guarantees to find the optimal alignment under the objective function, and otherwise is said to be near-optimal or approximate.

A given sequence is threaded through a given structure by searching for a sequence-structure alignment that places sequence amino acids into preferred structural environments and near other preferred amino acid types. The two key conditions that determine the complexity of this search [61] are whether or not (1) variable-length gaps are admitted into the alignment, and (2) interactions between neighboring amino acids from the sequence being threaded are admitted into the score function. Finding the optimal alignment is NP-hard in the general case where both conditions are allowed [61, 62], and low-order polynomial in the singleton-only case allowing variable-length gaps.

If variable-length gaps are not permitted, then alignments are restricted to substructures and subsequences of equal length that are extracted from a database. However, in a predictive setting the alignment method must allow alignment gaps to account for the loop length variability that is observed across structural families. Ignoring variable-length gaps means that the structure and a novel sequence almost invariably will be partially out of hydrophobic registration [11]. Consequently, this alternative is of little use for prediction.

If variable-length gaps are permitted but pairwise or higher-order interactions between sequence amino acids not allowed then the global optimum alignment can be found using the dynamic programming alignment method [21]. Dynamic programming alignment employs an affine gap penalty which biases the search to prefer loop lengths present in the core template's original sequence, and so would make distant structural homologs more difficult to recognize if their loop lengths differed substantially [41]. In addition, ignoring amino acid interactions means giving up a potentially rich source of structural information.

Alternatively, if both variable-length gaps and interactions between neighboring amino acids are allowed, then finding the global optimum threading is NP-hard [61]. This means that in order to find an optimal solution, any known algorithm must require an amount of time that is exponential in protein size. Consequently, approximate search algorithms are relatively fast and capable of finding a good but not necessarily the optimal solution, while exact search algorithms are guaranteed to find the optimal solution but sometimes require exponential time. Approximate alignment search methods include double dynamic programming, which employs a secondary level of dynamic programming to fix the neighbors for the first level [63, 64]; the "frozen approximation," a dynamic programming adaptation which substitutes the original motif residues initially and previous aligned sequence residues in subsequent steps [51]; and the Gibbs sampler, an iterative improvement method not based on dynamic programming which estimates a probability distribution and samples accordingly [65]. Exact methods include the branch and bound search described below [1], and exhaustive search [27].

2.3.4 core template selection or fold recognition (requirement iv)

Selecting a core template from the library (requirement iv) usually is approached by aligning the sequence to each member of the library and selecting the one yielding the best alignment score, usually after normalizing or correcting the raw scores in some way. Aligning a given sequence to several cores produces raw scores that usually are not directly comparable. Normalizing terms attempt to correct for biases due to search space size (larger search spaces have larger extreme tail values), core size (larger cores have more terms in the objective function), sequence composition, structural environment frequency, and so on. Score normalizations have included z-scores measured relative to near-optimal threadings of library structures, shuffled sequences of the same composition, random structures, or sequences of random composition; sequence composition corrections; accounting for the variable size of the core templates, or of the alignment search space; and reference states based on assumptions from statistical mechanics, mathematical statistics, or sampling theory.

2.4 Gapped block alignment

The gapped block alignment definition of threading used here follows Greer [17], Jones *et al.* [66], and Bryant & Lawrence [27]: (1) alignment gaps are prohibited within modeled secondary structure segments; (2) specific pairwise amino acid interactions are confined to the core template; and (3) loops are scored by an arbitrary sequence-specific function.

When a sequence is threaded through the core template, successive core elements of each segment are occupied by adjacent amino acids from the sequence. Note that conserved super-secondary structures such as beta-hairpins or tight turns could be included as additional types of core segments. No alignment gaps are permitted within segments, as this would correspond to an unphysical break in the sequence mainchain. Alignment gaps are confined to the connecting non-core loop regions, which undergo evolutionary insertions and deletions over time and are not usually considered part of the conserved core motif. Because the loops are viewed as variable, they do not participate in specific pairwise amino acid interactions under the objective function. The loop score function depends only on how the sequence is threaded through pairs of core segments; threading two adjacent core segments fixes the subsequence in the intervening loop region. Since loop endpoints are known, the sequence residues occupying a single loop in principle could be modeled in three-dimensional detail, but not modeled in general with other loops. More commonly, loop regions are treated simply as one or more additional structural environment types (e.g., tight turn, short, medium, and long), and objective function parameters are generated for them just as for any other singleton-only score term.

In contrast, with the use of an alignment method based on affine gap penalties, some threadings delete portions of the sequence or the structure. In analytical terms, this causes the global sum in equation 22 below to be over different effective structures and different effective sequence lengths. See Flöckner *et al.* [67] or Maiorov & Crippen [68] for cogent criticism of allowing parts of the sequence or structure to "vanish" in this way. In probabilistic terms, the usual linear or affine gap penalty forces loops to become exponentially unlikely in the length of the insertion or deletion. See Benner *et al.* [69] for empirical data showing that an exponential distribution does not provide an adequate fit to observed gap lengths. See Lemer *et al.* [36] for a discussion of inappropriate gap penalties, leading to gaps that are obviously far too small, as one aspect of threading algorithms that contributes to error.

Alternative methods that do use amino acid pair interaction terms directly in the loop score function require additional information to determine the loop placement in three-dimensional space. This information generally is not available in a predictive setting because the loop backbone coordinates often shift substantially to accommodate insertions or deletions. Maiorov & Crippen [68] propose an elegant approach that would include the needed three-dimensional information for pairwise terms. An open computational problem is to provide a practical algorithm for their scheme.

3 Formalization

The formulation below deliberately isolates the computational methods as much as possible from any particular theory of protein structure, from the way structural environments are defined, and from the score function employed. Consequently the methods apply to a wide variety of score functions that utilize pairwise amino acid interactions. Additional background on the problem formalization and notational conventions may be found in references [1, 3, 58, 59, 61], which this chapter follows where possible but revises where necessary. Table 1 summarizes the notation used.

Table 1 about here.

3.1 Sequence

The sequence **a** is a string of length n over an alphabet A of twenty characters (amino acid residue types). The set A^n consists of all strings over A of length n. The sequence **w** is a summation variable over A^n.

3.2 Core templates and library

The core template C is drawn from a library $\mathcal{L}$ of cores. Core template C is composed of m core segments C_i, each of length $c_i = |C_i|$ and representing a set of contiguous amino acid positions. Core segments are usually the pieces of conserved structure comprising the tightly packed internal protein core, and may correspond to the backbone trace of conserved secondary structure segments. Each segment C_i is composed of primitive core elements $C_{i,j}$, for $1 \leq j \leq c_i$. Each element $C_{i,j}$ corresponds to a spatial position that may be occupied by a residue from the sequence. For generality we make no restriction on segment length, and when $c_i = 1$ the segments may correspond to single amino acid residue positions.

For simplicity in the presentation we assume in this chapter that core segment length is fixed, even though [1] showed how this can lead to biological threading errors. Some important approaches [70, 71] treat core segment length as variable by adding residue positions to, or deleting them from, core segment endpoints. This would be modeled according to section 4.8 with additional parameters specifying each segment endpoint adjustment relative to the core template. Similarly, in this chapter we assume fixed core topology (i.e., segment rank order and direction). Variable topology, e.g. alternate arrangement of β-strands in a β-sheet, arises easily from core segment permutations or reversals. This would be modeled according to section 4.8 with additional parameters specifying the rank order and direction of each segment. Except for section 4.8, however, this chapter assumes fixed segment length and topology.

3.3 Loops

Core segments are connected by a set λ of loop regions. The loop regions might equally well be the "gaps" (in a dynamic programming alignment sense) used by some formulations. Loop λ_i connects segment C_i to C_{i+1}, N-terminal leader λ_0 precedes C_1, and C-terminal trailer λ_m follows C_m. Knowing the endpoints of λ_i is equivalent to knowing the threadings of C_i and C_{i+1}.

The length of loop λ_i is the variable l_i and its maximum (respectively minimum) length is l_i^{max} (respectively l_i^{min}). Unless stated otherwise, $l_i^{max} = +\infty$ and l_i^{min} = the minimum geometric spanning loop length (i.e., the minimum loop length capable of spanning the distance between the end of C_i and the beginning of C_{i+1}). Other values may reflect knowledge of additional constraints. For example, loops assigned length zero or one by the crystallographer usually reflect constrained "kinks" in the secondary structure which should be retained ($l_i^{max} = l_i^{min} = l_i^{native}$) or restricted ($l_i^{max} = 1$ and $l_i^{min} = 0$). As another example, Bryant & Lawrence [27] set l_i^{max} and l_i^{min} based on the maximum and minimum loop lengths in an aligned homologous family. To simplify notation we assume $l_0^{min} = l_m^{min} = 0$, i.e., the leader and trailer loops have zero minimum length.

3.4 Adjacency graph

An adjacency graph (also called interaction matrix, interaction graph, neighbor matrix, contact map, etc.) describes core element positions that are "neighbors." Positions are defined to be neighbors if they interact in the score function. The adjacency graph consists of a set $\mathcal{V}$ of vertices and a set $\mathcal{E}$ of edges. Each core element $C_{i,j}$ corresponds one-to-one to a graph vertex $v \in \mathcal{V}$. Consequently, the adjacency graph vertices merely relabel the core elements. Pairs of vertices u and v which interact in the score function (neighbors) are connected by a graph edge $e \in \mathcal{E}$, sometimes written $e = \{u, v\}$. Each vertex v and each edge e is labeled by an environment function s. The vertex (residue) environment labels, $s(v)$, may describe local factors such as degree of solvent exposure, local secondary structure type, and so forth. The edge environment labels, $s(e)$, may encode distance or contact between amino acids, the local environments at each end of the edge, and so forth. The edges are directed because the local environments at the edge head and tail may differ. The unaligned loop regions do not participate directly in the adjacency graph of pairwise relations.

3.5 A threading of a sequence into a core

A given alignment ("threading") of sequence $\mathbf{a}$ to core C associates each core element $C_{i,j}$ with exactly one amino acid from $\mathbf{a}$ (i.e. the core segments may not overlap). A legal threading is subject to the further constraints that successive amino acids in the sequence necessarily fall into successive core elements within each segment C_i (i.e. the core segments may not have internal gaps), that the core segments retain their original topological ordering (i.e. the threading does not permute or reverse segments), and that loop sizes are within the legal ranges (i.e. every loop region is long enough to span between its flanking segments). A sequence-structure alignment ("threading") of $\mathbf{a}$ into C is completely described by the primary sequence indices of the amino acids placed into the first element of each core segment. This results in a vector of m integers, denoted by $\mathbf{t}^a$ in absolute coordinates and $\mathbf{t}$ in relative coordinates. Each absolute coordinate t_i^a specifies the index in the sequence $\mathbf{a}$ that is aligned to the first element of the i^{th} core segment. That is, sequence residue $\mathbf{a}[t_i^a]$ occupies core element $C_{i,1}$.

For simpler notation [1] we generally replace absolute sequence coordinates $\mathbf{t}^a$ by relative coordinates $\mathbf{t}$, defined by $t_i = t_i^a - \sum_{j<i}(c_j + l_j^{min})$. Let $\tilde{n} = n + 1 - \sum_i(c_i + l_i^{min})$ and $\tilde{l}_i = l_i^{max} - l_i^{min}$. Then $t_i = 1$ corresponds to the lowest legal value of t_i^a and $t_i = \tilde{n}$ to the highest. Below, the absence of the superscript a will indicate relative coordinates.

The i^{th} loop length l_i and segment length c_i are related to $\mathbf{t}^a$ and $\mathbf{t}$ by $l_i = t_{i+1}^a - t_i^a - c_i = t_{i+1} - t_i + l_i^{min}$. Due to the minimum spanning loop length constraints, $1 + \sum_{j<i}(c_j + l_j^{min}) \leq t_i^a \leq n + 1 - \sum_{j\geq i}(c_j + l_j^{min})$. Due to core segment topological ordering constraints, $t_i^a + c_i + l_i^{min} \leq t_{i+1}^a \leq t_i^a + c_i + l_i^{max}$. In relative coordinates, the minimum loop length constraints simplify to $1 \leq t_i \leq \tilde{n}$ and the ordering constraints simplify to $t_i \leq t_{i+1} \leq t_i + \tilde{l}_i$.

Fictitious segments C_0 (respectively C_{m+1}) are fixed at the beginning (respectively end) of the sequence whenever it is convenient for indicated summations or recurrence limits. By convention, $c_0 = c_{m+1} = 0$, i.e., fictitious segments have zero length; and $t_0 = 1$ and $t_{m+1} = \tilde{n}$, i.e., they are fixed.

3.6 Sets of threadings

The set of threadings $\mathcal{T}[C, n, \mathbf{b}, \mathbf{d}]$ consists of all legal alignments of any sequence of length n to the core template C such that $b_i \leq t_i \leq d_i$ is satisfied. Where the entire search space is the intended set, we simplify the notation by writing $\mathcal{T}[C, n]$. The vector $\mathbf{x}$ is a summation variable over $\mathcal{T}[C, n]$.

Where the sequence and core are clear from context, we simplify the notation by writing $\mathcal{T}[\mathbf{b}, \mathbf{d}]$. The integers b_i and d_i define an interval, $[b_i, d_i]$, made up of the allowed sequence coordinates for core segment C_i. These m intervals may be represented compactly by two m-length vectors, $\mathbf{b}$ and $\mathbf{d}$ (mnemonic for "**B**egin" and "en**D**"). This allows us to represent all sets $\mathcal{T}[\mathbf{b}, \mathbf{d}]$ that have the particularly simple form of an m-dimensional axis-parallel hyper-rectangle whose two opposite corners are the vectors $\mathbf{b}$ and $\mathbf{d}$. The entire search space is represented by the hyper-rectangle $1 \leq t_i \leq \tilde{n}$. Thus $\mathcal{T}[C, n] = \mathcal{T}[\mathbf{1}, \tilde{\mathbf{n}}] = \{\mathbf{t} | 1 \leq t_i \leq \tilde{n}\}$.

The ability to represent and manipulate the search space directly allows for controlling the search. A list of several hyper-rectangles corresponds to the union of the sets they represent. If a particular list of hyper-rectangles is used to initialize the search in section 8.2, the subsequent search will examine only the corresponding threadings.

Each hyper-rectangle also contains a large number of illegal threadings that violate spacing or ordering constraints. Illegal threadings are always ignored. By convention, if $\mathbf{t}^{ill}$ is an illegal threading then $f(\mathbf{t}^{ill}) = +\infty$. Whenever we speak of a set of threadings we mean only the legal ones. Whenever search space sizes are computed, only legal threadings are counted.

3.7 Objective function

For a specific core motif C and protein sequence $\mathbf{a}$, the score of a candidate threading is defined to be the sum of the scores of the vertices and edges in the adjacency graph and the bulk composition of the loop regions. By analogy to energy minimization, lower scores are considered better. We assume the availability of an objective function f satisfying

$$P(\mathbf{a}|n, C, \mathbf{t}) \propto \exp\left(-f(\mathbf{a}, C, \mathbf{t})\right) \tag{1}$$

where $\mathbf{a}$ is a sequence of length n; C is a core template; $\mathbf{t}$ is a vector that specifies a sequence-structure alignment (a threading) and whose i^{th} component t_i specifies the alignment of core segment i; and $P(A|B)$ is the conditional probability of A given B. Where arguments are clear from context, we simplify the notation by omitting them, writing $f(i, t_i)$ to abbreviate $f(\mathbf{a}, C, i, t_i)$, and so on. The function f is the negative logarithm of an unnormalized conditional probability. It encodes the probability of observing sequence $\mathbf{a}$, given the alignment $\mathbf{t}$ to core C. For example, White *et al.* [58] and Stultz *et al.* [59] describe how to construct an objective function based on Markov Random Field (MRF) theory that satisfies equation 1.

Many published threading approaches are grounded in an underlying probabilistic objective function of this general nature. In practice, they may convert the underlying probabilistic objective function from a strict conditional probability to an odds-ratio relative to some assumed reference state, say $P(\mathbf{a}|n, C, \mathbf{t})/P_{ref}(\mathbf{a}|n, C, \mathbf{t})$. In contrast, the Bayesian analysis uses only the strict conditional probability, $P(\mathbf{a}|n, C, \mathbf{t})$. This is equivalent to setting $P_{ref}(\mathbf{a}|n, C, \mathbf{t}) = \mathit{constant}$ in some odds-ratio approaches. The reference state is derived from first principles of probability theory.

3.7.1 The fully general objective function

Given a specific protein sequence and a core template of m core segments, the fully general form of the score function is

$$\begin{aligned} f(\mathbf{t}) \quad = \quad & \sum_i g_1(i, t_i) + \sum_i \sum_{j>i} g_2(i, j, t_i, t_j) + \ldots \\ & + \sum_i \sum_{j>i} \ldots \sum_{l>k} g_m(i, j \ldots k, l, t_i, t_j \ldots t_k, t_l) \end{aligned} \tag{2}$$

where $i, j \ldots k, l$ index core segments and $t_i, t_j \ldots t_k, t_l$ give their relative positions in the sequence. The final sum is repeated over m indices representing all m core segments, and reflects amino acid interactions among all m core segments simultaneously.

In practice there is insufficient data to specify all the free parameters such a function would imply. Approaches differ in where they terminate the expansion. Profile-based methods employ only the g_1 term. Dynamic programming methods that do not allow pairwise amino acid interactions employ the g_1 term plus an affine gap penalty g_2 term of the form $g_2(i, i+1, t_i, t_{i+1}) = a + b|l_i - l_i^{native}|$. Methods that permit pairwise interactions, as here, employ a full g_2 term. Triplet interactions [51, 53] would require a g_3 term. A score function requiring g_4 or higher terms [72, 73, 74, 75] would arise in treatment of steric packing among multiple core segments, linked constraint equations on structural environments, detailed geometric or environment modeling, and so forth.

3.7.2 General pairwise interaction objective function

In the pairwise case, the score of a candidate threading is defined to be a function only of the sum of (1) a series of g_1 terms, each of which depends only on the threading of a single core segment, plus (2) a series of g_2 terms, each of which depends only on the threading of a pair of core segments:

$$f(\mathbf{t}) \quad = \quad \sum_i g_1(i, t_i) + \sum_i \sum_{j>i} g_2(i, j, t_i, t_j) \tag{3}$$

The functions g_1 and g_2 are the essential point of contact between the search algorithm and any particular choice of scoring function, neighbor relationships, or structural environments. The search algorithm is driven only by g_1 and g_2, regardless of how the score function assigns values to them.

In principle, every core element and every possible pair of elements could be assigned a unique structural environment encoding a different score table, and each loop region could assign a different score to every possible spanning subsequence. Consequently equation 3 is fully general for pair interactions. In most threading schemes, the score of a candidate threading is built up from the scores of the vertices and edges in the adjacency graph, and the sequence composition of the loop regions. Score functions that depend on separation in the sequence, proximity to the N- or C-terminal end of the sequence, or specialized identities of particular segments (e.g., including a regular expression pattern match based on known enzymatic function) are accommodated easily because the segment numbers (i, j) and segment indices (t_i, t_j) appear explicitly in the g_1 and g_2 argument lists. Other score components may be included provided they depend only on singleton or pairwise core segment threadings as shown.

3.7.3 A typical pairwise score function

Here we give an example of one way that a score function might be constructed. Details will vary with the particular score function and environment definitions chosen.

For any threading $\mathbf{t}$, let $f_{\mathrm{v}}(v, \mathbf{t})$ be the score assigned to core element or vertex v, $f_{\mathrm{e}}(\{u, v\}, \mathbf{t})$ the score assigned to interaction or edge $\{u, v\}$, and $f_l(\lambda_i, \mathbf{t})$ the score assigned to loop region λ_i. Then the total score of the threading is

$$f(\mathbf{t}) = \sum_{v \in \mathcal{V}} f_{\mathrm{v}}(v, \mathbf{t}) + \sum_{\{u,v\} \in \mathcal{E}} f_{\mathrm{e}}(\{u, v\}, \mathbf{t}) + \sum_{\lambda_i \in \lambda} f_l(\lambda_i, \mathbf{t}). \tag{4}$$

We can rewrite this as a function of threadings of pairs of core segments as follows.

$$
\begin{aligned}
f(\mathbf{t}) &= \sum_i \sum_{v \in C_i} f_{\mathrm{v}}(v, \mathbf{t}) + \sum_i f_l(\lambda_i, \mathbf{t}) \\
&\quad + \sum_i \sum_j \sum_{\substack{\{u,v\} \in \mathcal{E} \\ u \in C_i \\ v \in C_j}} f_{\mathrm{e}}(\{u, v\}, \mathbf{t}) && (5) \\
&= \sum_i \left[\sum_{v \in C_i} f_{\mathrm{v}}(v, \mathbf{t}) + \sum_{\substack{\{u,v\} \in \mathcal{E} \\ u,v \in C_i}} f_{\mathrm{e}}(\{u, v\}, \mathbf{t}) \right] + f_l(\lambda_0, \mathbf{t}) + f_l(\lambda_m, \mathbf{t}) \\
&\quad + \sum_i \left[f_l(\lambda_i, \mathbf{t}; 0 < i < m) + \sum_{j \neq i} \sum_{\substack{\{u,v\} \in \mathcal{E} \\ u \in C_i \\ v \in C_j}} f_{\mathrm{e}}(\{u, v\}, \mathbf{t}) \right] && (6) \\
&= \sum_i g_1(i, t_i) + \sum_i \sum_{j > i} g_2(i, j, t_i, t_j) && (7)
\end{aligned}
$$

The singleton terms, in g_1, include contributions from sources such as (in order of equation 6) individual core elements assigned to particular structural environments, pairwise interactions within a single core segment, and loop scores of the N- and C-terminal loop regions. The pairwise terms, in g_2, include contributions from sources such as (in order of equation 6) interior loop scores, and pairwise interactions between different core segments.

3.7.4 Computing g_1 and g_2 efficiently

Pre-computing g_1 and g_2 and storing them in arrays permits rapid evaluation of individual threadings as in equation 7, compared to their time-consuming *ab initio* evaluation as implied by equation 4. Storing g_1 requires $\mathcal{O}(m)$ arrays of size $\tilde{n}$, and storing g_2 requires $\mathcal{O}(m(m-1)/2)$ arrays of size $\tilde{n}(\tilde{n}+1)/2$, though in practice less storage is required because some core segment pairs do not interact.

3.7.5 Singleton-only objective function

In this section we define an objective function that allows an arbitrary sequence-specific score function for each segment or loop, but ignores all pairwise or higher order interactions between non-adjacent segments. We use f^1 to distinguish this singleton-only objective function from the general case, and Z^1 and μ^1 to distinguish corresponding global sums and means.

Let $f_s(i, t_i)$ be the score for occupying segment C_i by the substring of length c_i beginning at $\mathbf{a}[t_i]$, and let $f_l(i, t_i, t_{i+1})$ be the score for occupying loop λ_i by the substring of length $l_i = t_{i+1} - t_i + l_i^{min}$ beginning at $\mathbf{a}[t_i + c_i]$. If desired, pair interactions between segments C_i and C_{i+1} may be encoded in $f_l(i, t_i, t_{i+1})$ as well.

Assume that the threading score is the sum of the segment and loop scores separately.

$$
f^1(\mathbf{a}, C, \mathbf{t}) = \sum_{i=1}^{m} f_s(i, t_i) + \sum_{i=0}^{m} f_l(i, t_i, t_{i+1}) \tag{8}
$$

Functions h_s and h_l are the unnormalized probability functions corresponding to f_s and f_l.

$$h_s(i, t_i) = \exp(-f_s(i, t_i)) \quad (9)$$
$$h_l(i, t_i, t_{i+1}) = \exp(-f_l(i, t_i, t_{i+1})) \quad (10)$$

By convention, all illegal or out-of-range indices imply score $+\infty$ (infinitely bad) and probability zero; and $h_s(0, x) = h_s(m+1, x) = 1$, i.e., fictitious segments have zero score and unit probability; and $h_l(0, x, x) = h_l(m+1, x, x) = 1$, i.e., they have zero length.

Function H_s (respectively H_l) is the sum of h_s (respectively h_l) over all strings over A of length c_i (respectively l_i).

$$H_s(i, t_i) = \sum_{\mathbf{w} \in A^{c_i}} h_s(\mathbf{w}, C, i, t_i) \quad (11)$$
$$H_l(i, t_i, t_{i+1}) = \sum_{\mathbf{w} \in A^{l_i}} h_l(\mathbf{w}, C, i, t_i, t_{i+1}) \quad (12)$$

Function $h_\lambda(i, t_i, t_{i+1})$ is the sequence-independent probability of observing loop length $l_i = t_{i+1} - t_i + l_i^{min}$ at loop λ_i. The assumption that loop lengths are independent yields

$$P(\mathbf{t}|n, C) = \prod_{i=0}^{m} h_\lambda(i, t_i, t_{i+1}) \quad (13)$$

If uninformative priors are used, then $h_\lambda(i, t_i, t_{i+1}) = |\mathcal{T}[C, n]|^{-1/(m+1)}$ and the equation is exact. If an empirical loop length distribution [69] is used, then h_λ is taken from empirical tables; in this case the equation is approximate because $\sum_i \tilde{l}_i = \tilde{n} - 1$ so the assumption of loop length independence is violated, but it may yield a biologically more plausible result in some cases.

Recall that the relative coordinates shown must be converted to absolute coordinates to obtain an actual index into $\mathbf{a}$; specifically, add $\sum_{j<i}(c_j + l_j^{min})$ (respectively $\sum_{j<i+1}(c_j + l_j^{min})$) to the second argument t_i (respectively the third argument t_{i+1}) of f_s, f_l, h_s, h_l, h_λ, H_s, and H_l; and add $\sum_{j<i}(c_j + l_j^{min})$ to the relative coordinate y in $\mathbf{a}[y]$.

3.7.6 Per-residue singleton-only objective function

In many current proposals, the singleton-only f_s (respectively f_l) is specialized further to be the sum of the individual sequence residue scores at each element of the segment (respectively loop). Here we give a simple way to derive f_s, f_l, H_s, and H_l, in such proposals.

Let $s(C_{i,j})$ be the structural environment assigned to core element $C_{i,j}$ and $s(\lambda_i)$ be the structural environment assigned to loop λ_i. $s(C_{i,j})$ potentially reflects a different structural environment for each core element, as annotated by the theory of protein structure used. The loop structural environment $s(\lambda_i)$ might be used to divide loops into categories, e.g., tight, short, medium, and long; or all loops might be assigned to a single generic loop environment. Let $f_a^A(a', s)$ be the score assigned to amino acid residue type $a' \in A$ in environment s, and let $h_a^A(a', s) = \exp(-f_a^A(a', s))$.

$$
\begin{aligned}
f^A(\mathbf{a}, C, \mathbf{t}) &= \sum_{i=1}^{m} f_s^A(i, t_i) + \sum_{i=0}^{m} f_l^A(i, t_i, t_{i+1}) && (14) \\
f_s^A(i, t_i) &= \sum_{j=1}^{c_i} f_a^A(\mathbf{a}[t_i + j - 1], s(C_{i,j})) && (15) \\
f_l^A(i, t_i, t_{i+1}) &= \sum_{j=1}^{l_i} f_a^A(\mathbf{a}[t_i + c_i + j - 1], s(\lambda_i)) && (16) \\
h_s^A(i, t_i) &= \exp(-f_s^A(i, t_i)) = \prod_{j=1}^{c_i} h_a^A(\mathbf{a}[t_i + j - 1], s(C_{i,j})) && (17) \\
h_l^A(i, t_i, t_{i+1}) &= \exp(-f_l^A(i, t_i, t_{i+1})) = \prod_{j=1}^{l_i} h_a^A(\mathbf{a}[t_i + c_i + j - 1], s(\lambda_i)) && (18) \\
H_s^A(i, t_i) &= \prod_{j=1}^{c_i} \sum_{a' \in A} h_a^A(a', s(C_{i,j})) && (19) \\
H_l^A(i, t_i, t_{i+1}) &= \prod_{j=1}^{l_i} \sum_{a' \in A} h_a^A(a', s(\lambda_i)) && (20)
\end{aligned}
$$

4 Analysis — Selection tasks

For a given sequence, this section develops formulae for selecting

1. the most probable alignment $\mathbf{t}$ to a given core template, which maximizes $P(\mathbf{t}|\mathbf{a}, n, C)$ (equation 24 in section 24);

2. the most probable core template C across the entire library, which maximizes $P(C|\mathbf{a}, n)$ (equation 28 in section 28);

3. the most probable joint core template and alignment $\langle C, \mathbf{t}\rangle$ across the entire library, which maximizes $P(C, \mathbf{t}|\mathbf{a}, n)$ (equation 34 in section 34), and which need not be the most probable alignment of the most probable core template; and

4. the most probable core template segment alignments across the entire library, which maximize $P(C, i, t_i|\mathbf{a}, n)$ (equation 36 in section 36), and which may potentially allow for the construction of a core template for a sequence whose core template is not in the library by selecting piecewise the most probable segment alignments from different core templates.

Item 1 above corresponds to requirement (iii) in section 2.3, item 2 corresponds to requirement (iv), item 3 corresponds to (iii) and (iv) simultaneously, and item 4 corresponds to (iii) and (iv) for individual core segments. See figures 1–3.

4.1 Bayesian analysis

Bayes [76] provided the first exact treatment of inference based on inverting conditional probabilities. His interpretation of the formula, $P(A|B) = P(B|A)P(A)/P(B)$, is well known. Today the well-understood mathematics of Bayesian methods is a central component of optimal statistical inference [77, 78]. Conditional probability and Bayesian methods have been applied to protein secondary structure [59, 79], side-chain packing [80], fragment assembly [46], multiple sequence alignment [65], solvent exposure prediction [81], and structure classification [59, 82, 83], all with good results.

Here, the Bayesian analysis provides a compact account of the globally most probable cores and alignments. The Bayesian analysis presented here is closely related to other threading approaches as shown in figure 4. All approaches compute a transform from observed database frequencies to predicted probable structures. The Bayesian analysis is somewhat more direct and compact, but somewhat less biologically intuitive, than pseudo energy potential approaches.

Figure 4 about here.

4.1.1 Prior probabilities

Three prior probabilities are necessary for this analysis: $P(\mathbf{a}|n)$, $P(C|n)$, and $P(\mathbf{t}|n, C)$. $P(\mathbf{a}|n)$ is constant for a given sequence and may be ignored. $P(C|n)$ corresponds to the sequence-independent part of the core template probability. It reflects at least two influences: the relative frequencies of different core templates, and the way these shift with sequence length. $P(\mathbf{t}|n, C)$ corresponds to the sequence-independent part of the loop probability. It reflects the loop length probability distribution for C and n, independent of the specific amino acid residue types that actually occupy the loops. Assuming uninformative priors and fixed n, $P(C|n) = |\mathcal{L}|^{-1}$ and $P(\mathbf{t}|n, C) = |\mathcal{T}[C, n]|^{-1}$.

There are several plausible biological reasons why the assumption of uninformative priors might be relaxed. For example, $P(C|n)$ might instead reflect the observation that some folds are more probable than others [16, 24, 84]; or that fold-space attractors have unequal population densities [85]; or that proteins are roughly half secondary structure and half coil; or that longer sequences are more likely to fold into larger structures. $P(\mathbf{t}|n, C)$ might instead reflect an empirical loop length distribution [69] constructed by tabulating the loop lengths observed to connect loop endpoints in various geometries across a structural database; or a linear or affine gap penalty, in which case it becomes exponentially improbable in the number and length of insertions and deletions. This is not to argue for or against any particular set of priors. Rather, different informative priors might be plausible or not under different circumstances and assumptions.

4.1.2 Global sums

Only four global sums are sufficient to accomplish all of the probabilistic selections described above. It is easy to see that global sums and global means are equivalent, because if we know one we easily can produce the other. Below, Z denotes global sums and μ denotes global means. As with many biological problems where large search spaces and non-local influences co-occur, their computation is NP-hard if specific pair interactions and gaps are both permitted. Consequently either pruned search, approximations, or long computation must be employed. Section 7.3 provides brief proof sketches of NP-hardness for the general pair interaction case.

4.2 Selecting an alignment given a core template

For fixed sequence and core template, the task of sequence-structure alignment is to select an alignment of the sequence to the core template (see figure 2). White *et al.* [58] show that

$$P(\mathbf{a}|n, C, \mathbf{t}) = \frac{\exp(-f(\mathbf{a}, C, \mathbf{t}))}{Z_{\langle n,C,\mathbf{t}\rangle}} \tag{21}$$

$$Z_{\langle n,C,\mathbf{t}\rangle} = \sum_{\mathbf{w}\in A^n} \exp(-f(\mathbf{w}, C, \mathbf{t})) \tag{22}$$

$$P(\mathbf{t}|\mathbf{a}, n, C) = P(\mathbf{a}|n, C, \mathbf{t})\frac{P(\mathbf{t}|n, C)}{P(\mathbf{a}|n, C)} \tag{23}$$

$$= \frac{P(\mathbf{t}|n, C)}{P(\mathbf{a}|n, C)}\frac{\exp(-f(\mathbf{a}, C, \mathbf{t}))}{Z_{\langle n,C,\mathbf{t}\rangle}} \tag{24}$$

$Z_{\langle n,C,\mathbf{t}\rangle}$ is the global sum over all possible sequences of length n aligned by $\mathbf{t}$ to core template C. When f is modeled as a Markov Random Field, as in White *et al.* [58] and Stultz *et al.* [59], $Z_{\langle n,C,\mathbf{t}\rangle}$ is the same for every $\mathbf{t} \in \mathcal{T}[C, n]$. This case is treated here. In this case, assuming uninformative priors, the globally most probable alignment also is the one of globally lowest alignment score. Section 4.8 treats the case where $Z_{\langle n,C,\mathbf{t}\rangle}$ is allowed to vary with $\mathbf{t}$. In this case, even assuming uninformative priors, the globally most probable alignment may not have the globally lowest alignment score; the variability of $Z_{\langle n,C,\mathbf{t}\rangle}$ also must be accounted for.

4.3 Selecting a core template

For a fixed sequence, the task of fold recognition is to select a core template from the structure library (see figure 1). There is general agreement that one would like to select the core that has the highest conditional probability given the sequence.

$$P(C|\mathbf{a}, n) = \sum_{\mathbf{x}\in\mathcal{T}[C,n]} \frac{P(\mathbf{a}, n, C, \mathbf{x})}{P(\mathbf{a}, n)} \tag{25}$$

$$= \sum_{\mathbf{x}\in\mathcal{T}[C,n]} \frac{P(\mathbf{a}|n, C, \mathbf{x})P(\mathbf{x}|n, C)P(C|n)P(n)}{P(\mathbf{a}|n)P(n)} \tag{26}$$

$$= \frac{P(C|n)}{P(\mathbf{a}|n)} \sum_{\mathbf{x}\in\mathcal{T}[C,n]} \frac{\exp(-f(\mathbf{a}, C, \mathbf{x}))P(\mathbf{x}|n, C)}{Z_{\langle n,C,\mathbf{t}\rangle}} \tag{27}$$

$$= \frac{P(C|n)}{P(\mathbf{a}|n)} \frac{\mu_{\langle \mathbf{a},n,C \rangle}}{Z_{\langle n,C,\mathbf{t} \rangle}} \tag{28}$$

$$\mu_{\langle \mathbf{a},n,C \rangle} = \sum_{\mathbf{x} \in \mathcal{T}[C,n]} \exp(-f(\mathbf{a},C,\mathbf{x}))P(\mathbf{x}|n,C) \tag{29}$$

$$\mu_{\langle n,C,\mathbf{t} \rangle} = \sum_{\mathbf{w} \in A^n} \exp(-f(\mathbf{w},C,\mathbf{t}))P(\mathbf{w}|n,C) \tag{30}$$

Equation 28 orders all cores by conditional probability across the library.

Normalizing by $\sum_{C' \in \mathcal{L}} P(C'|\mathbf{a},n)$ imposes the fundamental threading assumption that the proper core is indeed in the library, and converts the ordering into a conditional probability reflecting that assumption. Similar normalizations reflecting the fundamental threading assumption apply throughout.

Assuming uninformative priors, the most probable core template maximizes the ratio $\mu_{\langle \mathbf{a},n,C \rangle}/Z_{\langle n,C,\mathbf{t} \rangle}$. Uninformative priors implies that $\mu_{\langle n,C,\mathbf{t} \rangle} = P(\mathbf{a}|n)Z_{\langle n,C,\mathbf{t} \rangle}$, in which case it also maximizes the ratio $\mu_{\langle \mathbf{a},n,C \rangle}/\mu_{\langle n,C,\mathbf{t} \rangle}$. This ratio is the mean probability across all possible alignments holding the sequence fixed, divided by the mean probability across all possible sequences holding the alignment fixed.

4.4 Selecting structure and alignment jointly

The central problem of inverse structure prediction is to select simultaneously both a core template and an alignment, given a sequence (see figure 3). To predict accurately, both the core template and the alignment must be selected correctly. However, it is evident by comparing equations 24 and 28 to equations 31 and 34 that the most probable alignment to the most probable core template is not necessarily equivalent to the most probable structure-alignment pair considered jointly.

$$P(C,\mathbf{t}|\mathbf{a},n) = P(\mathbf{t}|\mathbf{a},n,C)P(C|\mathbf{a},n) \tag{31}$$

$$= \frac{P(\mathbf{a}|n,C,\mathbf{t})P(\mathbf{t}|n,C)}{P(\mathbf{a}|n,C)} \frac{P(\mathbf{a}|n,C)P(C|n)}{P(\mathbf{a}|n)} \tag{32}$$

$$= \frac{P(\mathbf{a}|n,C,\mathbf{t})P(\mathbf{t}|n,C)P(C|n)}{P(\mathbf{a}|n)} \tag{33}$$

$$= \frac{P(C|n)P(\mathbf{t}|n,C)}{P(\mathbf{a}|n)} \frac{\exp(-f(\mathbf{a},C,\mathbf{t}))}{Z_{\langle n,C,\mathbf{t} \rangle}} \tag{34}$$

This orders all $\langle$structure, alignment$\rangle$ pairs by conditional probability jointly across the entire structure library.

4.5 Selecting individual core segment alignments

By selecting alignments to the most probable segments across the entire library, it might in principle be possible to construct a new core template piecewise out of the selected segments even though the constructed core template does not yet appear in the library. In this way it might in principle be possible to work around a current limitation of protein threading, namely that only known core structures may be predicted.

$$P(C, i, t_i|\mathbf{a}, n) = \sum_{\{\mathbf{x} \in \mathcal{T}[C,n] | x_i = t_i\}} P(C, \mathbf{x}|\mathbf{a}) \tag{35}$$

$$= \frac{P(C|n)}{P(\mathbf{a}|n)} \frac{\mu_{\langle \mathbf{a}, n, C, i, t_i \rangle}}{Z_{\langle n, C, \mathbf{t} \rangle}} \tag{36}$$

$$\mu_{\langle \mathbf{a}, n, C, i, t_i \rangle} = \sum_{\{\mathbf{x} \in \mathcal{T}[C,n] | x_i = t_i\}} \exp\left(-f(\mathbf{a}, C, \mathbf{x})\right) P(\mathbf{x}|n, C) \tag{37}$$

This orders all $\langle$structure, segment number, sequence index$\rangle$ triples by conditional probability across the entire library. The triples so generated (a) potentially arise from multiple different cores in the library, (b) have no overlap constraints between them, and (c) are selected from the set of legal threadings for each core, and so reflect its mean-field intra-template preferences and constraints. Furthermore, arranging for consistent pairwise or higher interactions between selected triples would result in a challenging constraint satisfaction problem. The problem of actually assembling such triples in a consistent manner into a novel "meta-core" is left open.

4.6 Super-secondary structures, or core template subsets

In many cases a core template may fit only partially to a structural analog. Some secondary structure segments may correspond, while others may not. This might be the case, for example, when a common super-secondary structure motif is shared but the rest of the protein diverges; or when part of the core superposes but another part does not. Suppose that k of the m segments correspond, that the corresponding segments are $I = \{i_1, i_2, \ldots, i_k\}$, and that the corresponding indices are $T = \{t_{i_1}, t_{i_2}, \ldots, t_{i_k}\}$. The previous section gave the special case when $k = 1$, and the caveats there apply here too.

$$P(C, I, T|\mathbf{a}, n) = \sum_{\{\mathbf{x} \in \mathcal{T}[C,n] | j \in I \Rightarrow x_j = t_j\}} P(C, \mathbf{x}|\mathbf{a}) \tag{38}$$

$$= \frac{P(C|n)}{P(\mathbf{a}|n)} \frac{\mu_{\langle \mathbf{a}, n, C, I, T \rangle}}{Z_{\langle n, C, \mathbf{t} \rangle}} \tag{39}$$

$$\mu_{\langle \mathbf{a}, n, C, I, T \rangle} = \sum_{\{\mathbf{x} \in \mathcal{T}[C,n] | j \in I \Rightarrow x_j = t_j\}} \exp\left(-f(\mathbf{a}, C, \mathbf{x})\right) P(\mathbf{x}|n, C) \tag{40}$$

This orders, by conditional probability across the entire library, all super-secondary structures or core template subsets that consist of k segments all taken from the same core template.

4.7 Secondary structure prediction

Let helix(j) denote the event that the j^{th} sequence residue $\mathbf{a}[j]$ is found in a helical conformation, and $\Phi_{helix}(\mathbf{a}, j, C, i) = \{t_i | \ C_i$ is helix and t_i places C_i over $\mathbf{a}[j]\}$; that is, the set of all threading indices for C_i that thread $a[j]$ to a helix position. Then

$$P(\text{helix}(j)|\mathbf{a}, n) = \sum_{C \in \mathcal{L}} P(\text{helix}(j)|\mathbf{a}, n, C) P(C|\mathbf{a}, n) \tag{41}$$

$$= \sum_{C\in\mathcal{L}} P(C|\mathbf{a},n) \sum_{\mathbf{t}\in\mathcal{T}[C,n]} P(\text{helix}(j)|\mathbf{a},n,C,\mathbf{t})P(\mathbf{t}|\mathbf{a},n,C) \tag{42}$$

$$= \sum_{C\in\mathcal{L}} P(C|\mathbf{a},n) \sum_{i=1}^{m} \sum_{t_i\in\Phi_{helix}(\mathbf{a},j,C,i)} P(C,i,t_i|\mathbf{a},n) \tag{43}$$

The final equation follows because for physical reasons $\mathbf{a}[j]$ cannot simultaneously be in two different helices or two different positions of the same helix. $P(\text{extended}(j)|\mathbf{a},n)$ is defined similarly (extended = β-sheet). For 3-state prediction, coil is defined as anything that is not helix or extended. Let $\Phi_{coil}(\mathbf{a},j,C,i) = \{\langle t_i, t_{i+1}\rangle |\ t_i$ places C_i before $\mathbf{a}[j]$ and t_{i+1} places C_{i+1} after it$\}$, where by convention t_0 = the beginning and t_{m+1} = the end of the sequence accounts for the leader and trailer loop regions. Then

$$P(\text{coil}(j)|\mathbf{a},n)$$
$$= \sum_{C\in\mathcal{L}} P(C|\mathbf{a},n) \sum_{i=0}^{m} \sum_{\langle t_i,t_{i+1}\rangle\in\Phi_{coil}(\mathbf{a},j,C,i)} P(C,\{i,i+1\},\{t_i,t_{i+1}\}|\mathbf{a},n) \tag{44}$$

The terms are given by equations 28, 36, and 39 with $k=2$, adjusted for boundary cases at sequence endpoints. As elsewhere, the values correspond to unnormalized probabilities.

4.8 Variable $Z_{\langle n,C,\mathbf{t}\rangle}$

The equations above treat the case where $Z_{\langle n,C,\mathbf{t}\rangle}$ is the same for every $\mathbf{t}\in\mathcal{T}[C,n]$. Here we treat the case where $Z_{\langle n,C,\mathbf{t}\rangle}$ varies with $\mathbf{t}$. This would occur with non-physical loop functions such as an affine gap penalty, or with variable segment length, rank order, connectedness, or topology. In this case, f induces a partition on $\mathcal{T}[C,n]$ such that two threadings $\mathbf{t}$ and $\mathbf{u}$ are in the same partition element if and only if $Z_{\langle n,C,\mathbf{t}\rangle} = Z_{\langle n,C,\mathbf{u}\rangle}$. Let the induced partition be $\mathcal{T}^*[C,n] = \{\mathcal{T}_i^*[C,n]\}$ where $\mathcal{T}_i^*[C,n] \subseteq \mathcal{T}[C,n]$ is the i^{th} partition element and the $\mathcal{T}_i^*[C,n]$ are disjoint and cover $\mathcal{T}[C,n]$. Let ${}_iZ$ (respectively ${}_i\mu$) represent global sums (respectively global means) over threadings in partition element $\mathcal{T}_i^*[C,n]$. Define

$${}_iZ_{\langle n,C,\mathbf{t}\rangle} = \sum_{\mathbf{w}\in A^n} \exp\left(-f(\mathbf{w},C,\mathbf{t})\right) \quad \text{, where } \mathbf{t}\in\mathcal{T}_i^*[C,n] \tag{45}$$

$${}_i\mu_{\langle\mathbf{a},n,C\rangle} = \sum_{\mathbf{x}\in\mathcal{T}_i^*[C,n]} \exp\left(-f(\mathbf{a},C,\mathbf{x})\right)P(\mathbf{x}|n,C) \tag{46}$$

$${}_i\mu_{\langle\mathbf{a},n,C,I,T\rangle} = \sum_{\{\mathbf{x}\in\mathcal{T}_i^*[C,n]|j\in I\Rightarrow x_j=t_j\}} \exp\left(-f(\mathbf{a},C,\mathbf{x})\right)P(\mathbf{x}|n,C) \tag{47}$$

Equation 28 must be generalized to

$$P(C|\mathbf{a},n) = \frac{P(C|n)}{P(\mathbf{a}|n)} \sum_{T_i^*[C,n]} \frac{{}_i\mu_{\langle\mathbf{a},n,C\rangle}}{{}_iZ_{\langle n,C,\mathbf{t}\rangle}} \tag{48}$$

Equation 39 must be generalized to

$$P(I,T,C|\mathbf{a},n) = \frac{P(C|n)}{P(\mathbf{a}|n)} \sum_{T_i^*[C,n]} \frac{{}_i\mu_{\langle\mathbf{a},n,C,I,T\rangle}}{{}_iZ_{\langle n,C,\mathbf{t}\rangle}} \tag{49}$$

Recall that equation 36 was the special case of equation 39 when $k = 1$, and must be generalized accordingly. Equations 24 and 29 are unchanged, but must be interpreted with variable $Z_{\langle n,C,\mathbf{t}\rangle}$.

4.9 Recurrence equations for singleton-only objective functions

Here we describe recursive formulae for the singleton-only objective function. This disallows pairwise interactions between sequence residues, but otherwise allows arbitrary sequence-specific score functions for the segments and loops and an arbitrary function for h_λ. We have implemented the relations below in Common LISP [86]. In practice we actually compute the logarithm of the quantities shown, then exponentiate only as needed, to avoid floating point problems.

4.9.1 Equations for $Z^1_{\langle n,C,\mathbf{t}\rangle}$

In the singleton-only case, if h_s and h_l are normalized probabilities, then $Z^1_{\langle n,C,\mathbf{t}\rangle} = H_s = H_l = 1$; otherwise, $Z^1_{\langle n,C,\mathbf{t}\rangle}$ corresponds to a normalizing constant for $\exp(-f^1)$.

$$Z^1_{\langle n,C,\mathbf{t}\rangle} = \sum_{\mathbf{w}\in A^n} \exp\left(-f^1(\mathbf{w},C,\mathbf{t})\right) \tag{50}$$

$$= \prod_{i=0}^{m} H_l(i,t_i,t_{i+1}) H_s(i,t_i) \tag{51}$$

4.9.2 Recurrence equations for $\mu^1_{\langle \mathbf{a},n,C\rangle}$

$$\mu^1_{\langle \mathbf{a},n,C\rangle} = \sum_{\mathbf{x}\in\mathcal{T}[C,n]} \exp\left(-f^1(\mathbf{a},C,\mathbf{x})\right) P(\mathbf{x}|n,C) \tag{52}$$

$$= \sum_{\mathbf{x}\in\mathcal{T}[C,n]} \prod_{i=0}^{m} h_l(i,x_i,x_{i+1}) h_\lambda(i,x_i,x_{i+1}) h_s(i+1,x_i) \tag{53}$$

Define an intermediate function R by the recurrence

$$R(m,x) = h_l(m,x,\tilde{n}) h_\lambda(m,x,\tilde{n}) \tag{54}$$

$$R(i,x) = \sum_{y=x}^{\tilde{n}} h_l(i,x,y) h_\lambda(i,x,y) h_s(i+1,y) R(i+1,y), \quad 0 \le i < m \tag{55}$$

$R(i,x)$ is the unnormalized probability corresponding to placing segment i at relative coordinate x but assigning it zero score, together with all following segments and loops, summed over all possible placements of the following segments. That is, $R(i,x)$ is $\mu^1_{\langle \mathbf{a},n,C\rangle}$ restricted to segments $i+1$ and above and the substring $\mathbf{a}[x]$ and beyond. Consequently,

$$\mu^1_{\langle \mathbf{a},n,C\rangle} = R(0,1) \tag{56}$$

4.9.3 Recurrence equations for $\mu^1_{\langle \mathbf{a},n,C,i,t_i\rangle}$

$$\begin{aligned}
\mu^1_{\langle \mathbf{a},n,C,i,t_i\rangle} &= \sum_{\{\mathbf{x}\in\mathcal{T}[C,n]|x_i=t_i\}} \exp\left(-f^1(\mathbf{a},C,\mathbf{x})\right)P(\mathbf{x}|n,C) && (57)\\
&= \sum_{\{\mathbf{x}\in\mathcal{T}[C,n]|x_i=t_i\}} \prod_{i=0}^{m} h_l(i,x_i,x_{i+1})h_\lambda(i,x_i,x_{i+1})h_s(i+1,x_i) && (58)\\
&= \sum_{\{\mathbf{x}\in\mathcal{T}[C,n]|x_i=t_i\}} \prod_{i=0}^{m} h_s(i,x_i)h_l(i,x_i,x_{i+1})h_\lambda(i,x_i,x_{i+1}) && (59)
\end{aligned}$$

where equation 59 follows because $h_s(0,x) = h_s(m+1,x) = 1$.

Define Q by the recurrence

$$\begin{aligned}
Q(1,x) &= h_l(0,1,x)h_\lambda(0,1,x) && (60)\\
Q(i,x) &= \sum_{y=1}^{x} Q(i-1,y)h_s(i-1,y)h_l(i-1,y,x)h_\lambda(i-1,y,x), && (61)\\
&\quad 1 < i \leq m+1
\end{aligned}$$

$Q(i,x)$ is the unnormalized probability corresponding to placing segment i at relative coordinate x but assigning it zero score, together with all preceding segments and loops, summed over all possible placements of the preceding segments. That is, $Q(i,x)$ is $\mu^1_{\langle \mathbf{a},n,C\rangle}$ restricted to segments $i-1$ and below and the substring $\mathbf{a}[x]$ and before. Consequently,

$$\mu^1_{\langle \mathbf{a},n,C,i,t_i\rangle} = Q(i,t_i)h_s(i,t_i)R(i,t_i) \tag{62}$$

4.9.4 Recurrence equations for $\mu^1_{\langle \mathbf{a},n,C,I,T\rangle}$

$$\begin{aligned}
\mu^1_{\langle \mathbf{a},n,C,I,T\rangle} &= \sum_{\{\mathbf{x}\in\mathcal{T}[C,n]|j\in I\Rightarrow x_j=t_j\}} \exp\left(-f^1(\mathbf{a},C,\mathbf{x})\right)P(\mathbf{x}|n,C) && (63)\\
&= \sum_{\{\mathbf{x}\in\mathcal{T}[C,n]|j\in I\Rightarrow x_j=t_j\}} \prod_{i=0}^{m} h_s(i,x_i)h_l(i,x_i,x_{i+1})h_\lambda(i,x_i,x_{i+1}) && (64)
\end{aligned}$$

Recall that $I = \{i_1, i_2, \ldots, i_k\}$ and $T = \{t_{i_1}, t_{i_2}, \ldots, t_{i_k}\}$. By convention, let $i_0 = 0$ and $t_{i_0} = 1$. Define Q_j by the recurrence

$$\begin{aligned}
Q_j(i_{j-1}+1,x) &= h_l(i_{j-1},t_{i_{j-1}},x)h_\lambda(i_{j-1},t_{i_{j-1}},x) && (65)\\
Q_j(i,x) &= \sum_{y=t_{i_{j-1}}}^{x} Q_j(i-1,y)h_s(i-1,y)h_l(i-1,y,x)h_\lambda(i-1,y,x), && (66)\\
&\quad i_{j-1}+1 < i \leq i_j
\end{aligned}$$

$Q_j(i,x)$ is the unnormalized probability corresponding to placing segment i at relative coordinate x but assigning it zero score, together with all preceding segments and loops back to but excluding placing segment i_{j-1} at $t_{i_{j-1}}$, summed over all possible placements of the intervening segments. Consequently, with $k = |I|$,

$$\mu^1_{\langle \mathbf{a},n,C,I,T\rangle} = \left(\prod_{j=1}^{k} Q_j(i_j, t_{i_j}) h_s(i_j, t_{i_j})\right) R(i_k, t_{i_k}) \tag{67}$$

For use with secondary structure prediction, loop modeling, α-/β-hairpins, etc., observe the special case of

$$\begin{aligned} &\mu_{\langle \mathbf{a},n,C,\{i,i+1\},\{t_i,t_{i+1}\}\rangle} \\ &\quad = Q(i,t_i)h_s(i,t_i)h_l(i,t_i,t_{i+1})h_\lambda(i,t_i,t_{i+1})h_s(i+1,t_{i+1})R(i+1,t_{i+1}) \end{aligned} \tag{68}$$

4.9.5 Recurrence equation invariants

Useful identities that may be used for diagnostic purposes include

$$\begin{aligned} \mu^1_{\langle \mathbf{a},n,C\rangle} &= \sum_{\mathbf{t}\in\mathcal{T}[C,n]} \exp\left(-f^1(\mathbf{a},C,\mathbf{t})\right) & (69)\\ &= R(0,1) & (70)\\ &= Q(m+1,\tilde{n}) & (71)\\ &= \sum_{t_i=1}^{\tilde{n}} \mu^1_{\langle \mathbf{a},n,C,i,t_i\rangle} & (72) \end{aligned}$$

In the interest of correct computer code, an implementation should verify equation 69 for every small search space, and the rest for every sequence–structure pair considered.

4.10 Recurrence equations for per-residue singleton-only objective functions

Many current proposals further specialize the singleton-only f_s (respectively f_l) to be the sum of the individual sequence residue scores at each element of the segment (respectively loop), as discussed in section 3.7.6. This leads to recurrence relations that are more efficient by a factor of $\tilde{n}$. However, the new recurrences no longer make loop endpoints or lengths explicit, so the uninformative loop prior $P(\mathbf{t}|n,C) = |\mathcal{T}[C,n]|^{-1}$ is assumed, a per-loop structural environment $s(\lambda_i)$ is used, and pair interactions between adjacent segments are not allowed. A superscript A indicates these assumptions.

The new recurrences are

$$\begin{aligned} R^A(m,x) &= h_l^A(m,x,\tilde{n})|\mathcal{T}[C,n]|^{-1} & (73)\\ Q^A(1,x) &= h_l^A(0,1,x) & (74)\\ Q_j^A(i_{j-1}+1,x) &= h_l^A(i_{j-1},t_{i_{j-1}},x) & (75) \end{aligned}$$

$$R^A(i,x) = \begin{cases} h_l^A(i,x,x)h_s^A(i+1,x)R^A(i+1,x) \\ \qquad +h_a^A(\mathbf{a}[x+c_i], s(\lambda_i))R^A(i,x+1), \\ \qquad 1 \leq x \leq \tilde{n} \text{ and } 0 \leq i < m \\ 0, \quad \text{otherwise} \end{cases} \tag{76}$$

$$Q^A(i,x) = \begin{cases} Q^A(i-1,x)h_s^A(i-1,x)h_l^A(i-1,x,x) \\ \qquad +Q^A(i,x-1)h_a^A(\mathbf{a}[x-1], s(\lambda_{i-1})), \\ \qquad 1 \leq x \leq \tilde{n} \text{ and } 1 < i \leq m+1 \\ 0, \quad \text{otherwise} \end{cases} \tag{77}$$

$$Q_j^A(i,x) = \begin{cases} Q_j^A(i-1,x)h_s^A(i-1,x)h_l^A(i-1,x,x) \\ \qquad +Q_j^A(i,x-1)h_a^A(\mathbf{a}[x-1], s(\lambda_{i-1})), \\ \qquad t_{i_{j-1}} \leq x \leq t_{i_j} \text{ and } i_{j-1}+1 < i \leq i_j \\ 0, \quad \text{otherwise} \end{cases} \tag{78}$$

Consequently,

$$Z^A_{\langle n,C,\mathbf{t}\rangle} = \prod_{i=0}^{m} H_l^A(i,t_i,t_{i+1})H_s^A(i,t_i) \tag{79}$$

$$\mu^A_{\langle \mathbf{a},n,C\rangle} = R^A(0,1) \tag{80}$$

$$\mu^A_{\langle \mathbf{a},n,C,i,t_i\rangle} = Q^A(i,t_i)h_s^A(i,t_i)R^A(i,t_i) \tag{81}$$

$$\mu^A_{\langle \mathbf{a},n,C,I,T\rangle} = \left(\prod_{j=1}^{k} Q_j^A(i_j,t_{i_j})h_s^A(i_j,t_{i_j})\right)R^A(i_k,t_{i_k}) \tag{82}$$

5 Analysis — Search space tasks

The branch and bound search algorithm, described in detail in section 8.1, works by repeatedly subdividing the search space into smaller subsets, always choosing the most promising subset to split at each step. In order to succeed, branch and bound search here requires the ability to (1) represent the entire search space as a set of possibilities; (2) split any set into subsets; and (3) compute a lower bound on the best score achievable within any subset. Any correct implementation of these three requirements would result in a correct search, but search speed would vary dramatically. The keys to an efficient search are a powerful lower bound and good branch points when splitting sets.

For a given sequence and a specific selected core template, this section develops formulae for lower bound, search space splitting, and various search space parameters.

5.1 Lower bound on scores in threading sets

The branch and bound search exploits a lower bound on the score $f(\mathbf{t})$ attainable by any threading $\mathbf{t}$ in any set $\mathcal{T}[\mathbf{b},\mathbf{d}]$. Any correct lower bound would result in correct search behavior, but the stronger the lower bound, the more rapidly the search prunes unwanted sets of threadings and converges to the optimum. Total search time is an engineering trade-off between a polynomial-time lower bound computation and an exponential-time search. As a general rule, stronger lower bounds are more expensive to compute but result in smaller exponent coefficients than do weaker lower bounds. Evaluation of the lower bound occurs in the inner loop of the search algorithm and consumes virtually all of its computation time. Consequently, it is crucial that it be computable efficiently.

For example, one lower bound that is easy to derive and fast to compute can be obtained from equation 3 by summing lower bounds on each term separately

$$\begin{aligned} \min_{\mathbf{t}\in\mathcal{T}} f(\mathbf{t}) &= \min_{\mathbf{t}\in\mathcal{T}} \sum_i \left[g_1(i,t_i) + \sum_{j>i} g_2(i,j,t_i,t_j) \right] \\ &\geq \sum_i \left[\min_{b_i\leq x\leq d_i} g_1(i,x) + \sum_{j>i} \min_{\substack{b_i\leq y\leq d_i \\ b_j\leq z\leq d_j}} g_2(i,j,y,z) \right] \end{aligned} \tag{83}$$

The indicated min operations are computable efficiently using binary trees over sub-intervals of $g_1(i,x)$, and quad-trees or 2D-trees [87] over sub-intervals of $g_2(i,j,y,z)$. This simple formula works well for small cases, and consequently would be useful for threading small super-secondary structure motifs or for testing a prototype branch and bound search implementation. It is sufficiently powerful to provide effective pruning in search space sizes of about 10^9 or 10^{12}.

We have explored several alternative forms of the lower bound [1]. Our current version, denoted $\mathrm{lb}(\mathcal{T})$, is effective in search space sizes up to about 10^{25} or 10^{30}.

$$\begin{aligned} \min_{\mathbf{t}\in\mathcal{T}} f(\mathbf{t}) &\geq \mathrm{lb}(\mathcal{T}) \\ &= \min_{\mathbf{t}\in\mathcal{T}} \sum_i \Bigg[g_1(i,t_i) + g_2(i-1,i,t_{i-1},t_i) \\ &\qquad\qquad + \min_{\substack{\mathbf{u}\in\mathcal{T} \\ l_j^{max}=+\infty}} \sum_{|j-i|>1} \frac{1}{2} g_2(i,j,t_i,u_j) \Bigg] \end{aligned} \tag{84}$$

The enclosing $\min_{\mathbf{t}\in\mathcal{T}}$ ensures that the lower bound will be instantiated on a specific legal threading $\mathbf{t}^{lb}\in\mathcal{T}$. This will be used in splitting $\mathcal{T}$, below. The equation further ensures that the singleton term, in $g_1(i,t_i)$, remains consistent both with the terms that reflect loop scores, in $g_2(i-1,i,t_{i-1},t_i)$, and with the other (non-loop) pairwise terms, in $g_2(i,j,t_i,u_j)$. The inner $\min_{\mathbf{u}\in\mathcal{T}}$ allows a different vector $\mathbf{u}$ for each i, but requires $\mathbf{u}$ to be a legal threading. The assumption $l_j^{max}=+\infty$ supports an efficient implementation. Equation 84 would be a tight lower bound (i.e, actually achieved in $\mathcal{T}$) if we further required that $\mathbf{u}=\mathbf{t}$; but then evaluating the bound would be equivalent to solving the search problem. It is easy to see that if $\mathcal{T}$ is a singleton set, $\{\mathbf{t}\}$, then $\mathrm{lb}(\{\mathbf{t}\}) = f(\mathbf{t})$.

5.1.1 Lower bound invariants

Two useful invariants are (1) $\mathcal{T}_1 \supset \mathcal{T}_2$ implies $\mathrm{lb}(\mathcal{T}_1) \leq \mathrm{lb}(\mathcal{T}_2)$, and (2) for any $\mathbf{t}$, $\mathrm{lb}(\{\mathbf{t}\})$ by equation 84, $f(\mathbf{t})$ by summing g_1 and g_2 by equation 3, and $f(\mathbf{t})$ by summing vertex, edge, and loop components by equation 7, all are equal. In the interest of correct computer code, an implementation should verify the first invariant whenever a subset is split, and the second whenever a global optimum threading is found.

5.2 Splitting threading sets

The second key element of branch and bound search as used here is the ability to subdivide sets of threadings successively. A set is split by choosing a single core segment C_i and a single split-point t_i^{split} (see figure 5). The interval $[b_i, d_i]$ is divided into three sub-intervals: the points (1) less than the split-point, $\left[b_i, t_i^{split} - 1\right]$; (2) equal to the split-point, $\left[t_i^{split}, t_i^{split}\right]$; and (3) greater than the split-point, $\left[t_i^{split} + 1, d_i\right]$. This results in three mutually disjoint and exhaustive subsets of the original set. There are many possible ways to choose C_i and t_i^{split}. The choice affects search speed, but not so much as does the choice of lower bound.

Currently, we choose t_i^{split} based on $\mathbf{t}^{lb} \in \mathcal{T}$. The specific threading t_i^{lb} that instantiates the lower bound in equation 84 is used to choose $t_i^{split} = t_i^{lb}$.

It is less obvious how to select the core segment C_i at which to split. One simple method, easy to implement and appropriate for threading small super-secondary structure motifs, is to split at the segment having the widest interval, i.e., at the i that maximizes the value of $d_i - b_i$. The method we currently use chooses the segment i that has the most negative expected score contribution if its interval were to be split at t_i^{lb}. Specifically, we split at $\langle i, t_i^{lb}\rangle$ where i yields the most negative value of the expression

$$P_1(i, t_i^{lb}) \left[g_1(i, t_i^{lb}) - \overline{g}_i + \sum_{j \neq i} \Big(1 - \alpha(j)/2\Big) \Big(g_2(i, j, t_i^{lb}, t_j^{lb}) - \overline{g}_{i,j} \Big) \right] \tag{85}$$

Assuming a uniform probability distribution over all legal threadings, $P_1(i, t_i^{lb})$ is the probability that a randomly drawn threading will place C_i at t_i^{lb}; $\overline{g}_i$ is the expected value of $g_1(i, *)$; $\overline{g}_{i,j}$ is the expected value of $g_2(i, j, *, *)$; these terms are defined in section 6. $\alpha(i)$ indicates whether segment i is active (variable) or inactive (fixed) in the set

$$\alpha(i) = \begin{cases} 1, & \text{if } b_i < d_i; \\ 0, & \text{if } b_i = d_i. \end{cases} \tag{86}$$

In equation 85, the factor $P_1(i, t_i^{lb})$ biases the choice to prefer combinations of i and t_i^{lb} that are *a priori* more likely. The terms $g_1(i, t_i^{lb}) - \overline{g}_i$ and $g_2(i, j, t_i^{lb}, t_j^{lb}) - \overline{g}_{i,j}$ bias the choice to prefer scores that are lower than expected. The factor $(1 - \alpha(j)/2)$ assigns the entire pairwise term $g_2(i, j, t_i^{lb}, t_j^{lb}) - \overline{g}_{i,j}$ to core segment C_i if C_j is inactive, and shares it evenly between them if both are active.

One-step look-ahead yields a much more effective, but much more expensive, heuristic for splitting sets. In this method, all m possible splitting segments are considered in turn. For each segment, three lower bounds are computed corresponding to its three resulting split sets ($<$, $=$, and $>$). Segments are ranked by the minimum lower bound among their three sets. The segment that maximizes the minimum lower bound is chosen as the splitting segment because it will result in the largest overall increase in lower bound. Because $3m$ lower bound computations must be done this approach is quite slow on a serial computer. In a distributed processing environment each lower bound could be computed simultaneously on a different processor, resulting in enhanced performance for the same elapsed search time.

6 Analysis — Search Space Formulae

Formulae in this section are used when splitting sets of threadings, and also to obtain important statistical information.

6.1 Fast approximate formulae

For the heuristic choice of which core segment to split in equation 85, speed is important and approximate formulae are acceptable. Fast formulae result from the simplifying assumptions that the entire search space is included ($b_i = 1$ and $d_i = \tilde{n}$), and that loops can be arbitrarily long ($l_i^{max} = +\infty$). Equations 87–89 are exact for this case, and are used to approximate all other cases in equation 85. Exact algorithms for all other cases are given below in sections 6.2 and 6.3.

Under these simplifying assumptions the number of legal threadings, or search space size, $S = |\mathcal{T}[C, n]|$, is the result of the binomial coefficient function

$$S \approx \binom{\tilde{n} + m - 1}{m}. \tag{87}$$

Simple formulae also hold for segment placement probabilities. Under a uniform probability distribution on threadings, let $P_1(i, t_i)$ be the probability that segment i occurs at index t_i in a randomly drawn threading, and let $P_2(i, j, t_i, t_j)$ be the probability that segment i occurs at index t_i and simultaneously segment j occurs at index t_j. Let $i < j$ and $t_i \leq t_j$. Then,

$$P_1(i, t_i) \approx \binom{t_i + i - 2}{i - 1}\binom{\tilde{n} - t_i + m - i}{m - i} \Big/ \binom{\tilde{n} + m - 1}{m} \tag{88}$$

$$P_2(i, j, t_i, t_j) \approx \binom{t_i + i - 2}{i - 1}\binom{t_j - t_i + j - i - 1}{j - i - 1}\binom{\tilde{n} - t_j + m - j}{m - j} \Big/ \binom{\tilde{n} + m - 1}{m} \tag{89}$$

Each factor is always the binomial coefficient corresponding to the number of ways to choose the number of available core segments, out of one less that the number of available sequence indices plus the number of available core segments. The denominator is the approximate search space size from equation 87. Successive factors in the numerator correspond to the combinatorial number of arrangements between successive pairs of core segments fixed by the arguments. Similar formulae hold for $P_3(i, j, k, t_i, t_j, t_k)$, for $P_4(i, j, k, l, t_i, t_j, t_k, t_l)$, and so forth.

These relations permit us to estimate the expected singleton and pairwise score components attributable to each segment. The expected singleton contribution for segment i is $\overline{g}_i$, and the expected pairwise contribution for segments i and j is $\overline{g}_{ij}$. Where $i < j$ and $t_i \leq t_j$,

$$\overline{g}_i = \sum_x P_1(i, x) g_1(i, x) \tag{90}$$

$$\overline{g}_{ij} = \sum_x \sum_{y \geq x} P_2(i, j, x, y) g_2(i, j, x, y) \tag{91}$$

6.1.1 Computing P_1 and P_2 efficiently

In practice we compute the logarithm of equations 88 and 89, then exponentiate. When loading the system we precompute and store $\log n$ for $n < 1,000$ and $\log \binom{n}{m}$ for $m < 50$ and $n < 1,000$. Equations 88 and 89 then require only the sum of a few array references plus a transcendental function call. The approximations to $P_1(i,t_i)$, $P_2(i,j,t_i,t_j)$, $\overline{g}_i$, and $\overline{g}_{ij}$, all are constant for a given search space, and are precomputed and stored when each search is begun. The precomputation is fast and the storage required is approximately the size of the g_1 and g_2 arrays. Consequently, equation 85 requires only sums and products of a few array references.

6.2 Exact search space size, probabilities, and uniform sampling

In practice, external knowledge may constrain core segments to arbitrary sub-intervals or specify maximum loop lengths. This section provides exact formulae for such cases.

6.2.1 Search space size

Let $\mathcal{T}[\mathbf{b},\mathbf{d}] = \{\mathbf{t} | b_i \le t_i \le d_i\}$ be the set of threadings delimited by $\mathbf{b}$ and $\mathbf{d}$, let $S[\mathbf{b},\mathbf{d}]$ be the number of legal threadings it contains, and let $B(i,x)$ be the number of legal threadings of segments i through m when segment i is placed at relative sequence index x or higher. B is given by the recursive formula

$$B(i,x) = \begin{cases} d_m - x + 1, & \text{if } i = m \text{ and } b_m \le x \le d_m; \\ B(i,b_i), & \text{if } 1 \le i \le m \text{ and } x < b_i; \\ B(i,x+1) + B(i+1,x) - B(i+1, x+\tilde{l}_i+1), & \\ & \text{if } 1 \le i < m \text{ and } b_i \le x < d_i; \\ 0, & \text{otherwise.} \end{cases} \tag{92}$$

The numbers involved in computing B become combinatorically large; arbitrary precision integer arithmetic is a language primitive in LISP, and usually available as a subroutine in other languages.

Consequently,

$$S[\mathbf{b},\mathbf{d}] = B(1,b_1) \tag{93}$$

is exact for arbitrary $\mathbf{b}$, $\mathbf{d}$, $\mathbf{l}^{min}$, and $\mathbf{l}^{max}$. By applying equation 92 to $b_i = 1$ and $d_i = \tilde{n}$,

$$S = S[\mathbf{1},\tilde{\mathbf{n}}] \tag{94}$$

gives the exact size of the entire legal search space. This is the exact formula corresponding to the approximate equation 87, and is used for all search space sizes reported here.

6.2.2 Exact segment placement probabilities

Exact formulae for segment placement probabilities are computable as the ratio of the search space sizes corresponding to the constrained and the entire search spaces. The denominator in all cases is the entire search space size given by equation 94. The numerator corresponding to $P_1(i,t_i)$ arises from the set of threadings that fix C_i at t_i, denoted $\mathcal{T}\langle i,t_i\rangle$. Its search space size $S\langle i, t_i \rangle$ may be computed from equation 92 applied to $b_j = \{\text{if } j < i \text{ then } 1 \text{ else } t_i\}$ and $d_j = \{\text{if } j \leq i \text{ then } t_i \text{ else } \tilde{n}\}$. Then

$$P_1(i, t_i) = S\langle i, t_i \rangle / S[\mathbf{1}, \tilde{\mathbf{n}}] \tag{95}$$

is the exact formula corresponding to the approximate equation 88. Similar formulae hold for $P_2(i, j, t_i, t_j)$, $P_3(i, j, k, t_i, t_j, t_k)$, $P_4(i, j, k, l, t_i, t_j, t_k, t_l)$, and so forth.

6.2.3 Uniform random sampling

Equation 92 also allows us to randomly sample the threadings in any set $\mathcal{T}[\mathbf{b}, \mathbf{d}]$, assuming a uniform probability distribution (blind draw) on threadings. Let s be a random integer uniformly drawn between one and $S[\mathbf{b}, \mathbf{d}]$ inclusive; uniform random numbers are a language primitive in LISP, and usually available as a subroutine in other languages. Convert s to a unique threading as follows:

FOR i FROM 1 TO m DO

1. Find x such that $b_i \leq x \leq d_i$ and $B(i, x+1) < s \leq B(i, x)$.
2. Set t_i to x.
3. Set s to $s - B(i, x+1)$.

It is only necessary to compute $S[\mathbf{b}, \mathbf{d}]$ and B once for each set $\mathcal{T}[\mathbf{b}, \mathbf{d}]$.

6.3 Exact analytic search space mean and standard deviation

Let $f(t) = \sum_i g_1(i, t_i) + \sum_i \sum_{i<j} g_2(i, j, t_i, t_j)$ be the threading score function chosen. Then the score distribution mean $\overline{f}$ is

$$\overline{f} = E(f(*)) = \sum_i E(g_1(i, *)) + \sum_i \sum_{i<j} E(g_2(i, j, *, *) \tag{96}$$

where

$$E(g_1(i, *)) = \sum_x P_1(i, x) g_1(i, x) \tag{97}$$

$$E(g_2(i, j, *, *)) = \sum_x \sum_y P_2(i, j, x, y) g_2(i, j, x, y) \tag{98}$$

The distribution variance is V and the standard deviation is $\sigma = \sqrt{V}$.

$$V = E([\overline{f} - f(*)]^2) = E(\overline{f}^2 - 2\overline{f} f(*) + f^2(*)) = E(f^2(*)) - \overline{f}^2 \tag{99}$$

where

$$E(f^2(*)) = E([\sum_i g_1(i, *) + \sum_i \sum_{i<j} g_2(i, j, *, *)]^2) \tag{100}$$

$$\begin{aligned}
&= \sum_i E([g_1(i,*)]^2) + 2\sum_i \sum_{i<j} E(g_1(i,*)g_1(j,*)) \\
&\quad + \sum_i \sum_{i<j} \sum_k E(g_1(k,*)g_2(i,j,*,*)) \\
&\quad + \sum_i \sum_{i<j} E([g_2(i,j,*,*)]^2) \\
&\quad +2\sum_i \sum_{i<j} \sum_{j<k} \sum_{k<l} E(g_2(i,j,*,*)g_2(k,l,*,*)) \qquad (101)
\end{aligned}$$

$$E([g_1(i,*)]^2) = \sum_x P_1(i,x)[g_1(i,x)]^2 \qquad (102)$$

$$E(g_1(i,*)g_1(j,*)) = \sum_x \sum_y P_2(i,j,x,y)g_1(i,x)g_1(j,y) \qquad (103)$$

$$E(g_1(k,*)g_2(i,j,*,*)) = \sum_x \sum_y \sum_z P_3(i,j,k,x,y,z)g_1(k,z)g_2(i,j,x,y) \qquad (104)$$

$$E([g_2(i,j,*,*)]^2) = \sum_x \sum_y P_2(i,j,x,y)[g_2(i,j,x,y)]]^2 \qquad (105)$$

$$\begin{aligned}
E(g_2(i,j,*,*)g_2(k,l,*,*)) &= \sum_x \sum_y \sum_z \sum_v P_4(i,j,k,l,x,y,z,v) \\
&\quad * g_2(i,j,x,y)g_2(k,l,z,v) \qquad (106)
\end{aligned}$$

The analytic formula for the mean has a computational complexity of $\mathcal{O}(m^2\tilde{n}^2)$. The analytic formula for the standard deviation has a computational complexity of $\mathcal{O}(m^4\tilde{n}^4)$.

6.4 Computing the exact analytic mean and standard deviation efficiently

In practice, the fourth-power computational complexity of the analytic standard deviation formula is burdensome for most proteins. Consequently, we usually estimate the mean and standard deviation by sampling the search space. It takes only a few seconds to draw and score 10,000 uniformly distributed random threadings using the g_1 and g_2 arrays in conjunction with methods in section 6.2.3. This results in sufficiently accurate estimates for most ordinary purposes. In case an exact value is important, the analytic formulae are available at additional computational cost.

7 Computational Complexity

The complexity of finding the optimal sequence-to-structure threading is a function of whether arbitrary length alignment gaps are allowed, and whether or not the score function includes pairwise or higher order interactions among sequence amino acids. Given these two properties, the threading problem is known to be NP-complete.

Our current knowledge of protein structure and evolution requires the first property, the allowance of alignment gaps of near arbitrary length. In fact, a very wide range of sequences have been observed to have been inserted into various basic folds, particularly at the protein's surface. The extreme case is observed in many multidomain proteins where the linear sequence encoding for one domain's fold has been inserted into that encoding a second independent fold.

The necessity of the second property, inclusion of at least long-range pairwise interactions in the scoring schema, is much less clear. It generally is believed to be important, but the issue is far from settled. In the three-dimensional folded form of a native protein, the key characteristics of the local environment in which any amino acid finds itself are determined largely by its contacting neighbors. Interactions between spatially neighboring amino acids are formed by those at arbitrary positions along the sequence.

The general problems of protein folding [88, 89, 90], protein threading [61], [62], and protein structure comparison [85], all are known to be NP-hard. Thus an exact solution is widely believed to require exponential time (unless $P = NP$). In the general case, any approach to the information transform shown in figure 4 must solve (or approximate) at least one NP-hard problem. Different approaches trade off which NP-hard problems to solve or approximate, and how. We expect that many current approaches will be found to contain computational analogs or approximations to the quantities above. Fast approximate knowledge-intensive solutions often perform well in practice, and a clear understanding of the optimal formulae can help us to evaluate the speed/accuracy trade-offs such approximate solutions must make.

7.1 Computational complexity and NP-completeness

An exhaustive review of computational complexity and NP-completeness is beyond the scope of this chaper, and this section presents only enough material to motivate the discussion in the balance of the section. The interested reader is referred to references [91, 92, 93] for formal treatment of the subject; and to references [88, 89, 90] for discussion of its biological relevance.

The analysis of computational complexity is concerned with the question of how the running time of an algorithm grows as the size of its input increases. For a given algorithm, this is made specific by naming some function, f, and asserting that the algorithm's running time grows with inputs of increasing size "no faster" than f grows with arguments of increasing magnitude. Formally, let $\#(A, I)$ denote the running time (number of steps) of algorithm A when started with input I, and let $|I|$ be some reasonable measure of the size of I. Then we write $A = \mathcal{O}(f)$, read "A is order of f" (or "A is big oh of f"), if there exists any positive constant C such that

$$\lim_{|I|\to\infty} \frac{\#(A, I)}{f(|I|)} \leq C$$

Algorithms whose computational complexity is the order of some polynomial ("polynomial time algorithms") are considered to be formally efficient. All other algorithms have a running time that grows faster than any possible polynomial, and are considered to be inefficient. It is possible, of course, that an exponential time algorithm with a tiny exponent may terminate rapidly on small and medium sized inputs, or that a polynomial time algorithm applied to a very large input may not. Nonetheless, the distinction is valuable and important in most practical cases.

A "problem" is a class of computational tasks defined in terms of a set of parameters (for example, SEQUENCE is a parameter of the protein threading problem). A problem "instance" results from replacing the parameters by actual values (for example, replacing SEQUENCE by a specific string). An algorithm solves a problem if it terminates correctly on every instance of the problem. A "decision problem" is one to which the answer is either "YES" or "NO." For example, the decision problem addressed in protein threading is, "Does there exist a threading of this sequence into this structure under this score function, such that the threading score is less than K?" This might be the case in which a candidate threading already has been found, and one wishes simply to ask whether or not another threading with a better score exists.

It is customary to identify the computational complexity of a problem with that of the most efficient algorithm that solves it. The class of problems that can be solved by a polynomial time algorithm is named "P." Problems which belong to this class are formally tractable. The class of decision problems whose solution, *once found or guessed,* may be verified in polynomial time is named "NP." Note that it may not be possible actually to *find* a solution in polynomial time; the condition refers only to the *verification* of a putative solution. Problems in NP are solvable in "non-deterministic polynomial time," meaning that if one could somehow non-deterministically guess and check all possible solutions simultaneously, the solution would be obtained in polynomial time. The practical message of this theoretical condition is that the search for a solution, not the verification step, determines whether a polynomial time solution is possible or not. Clearly, P is a subset of NP, but it is unknown whether P=NP.

There is an important class of problems which belong to NP, and have the property that an algorithm solving any problem in the class can be transformed in polynomial time to solve any other problem in NP. Therefore, a polynomial time algorithm solving any problem in this class would immediately yield a polynomial time solution for every problem in NP. These problems are the hardest in NP, and are known as the "NP-complete" problems. It is not known whether or not they have a polynomial time solution (if so, then P=NP because polynomials are closed under composition). They include many problems deeply central to computer science, and so a great deal of effort by a great many talented people has been expended searching for a polynomial time solution to any one of them. Because so many talented people have failed, it is widely accepted that no polynomial time algorithm is likely ever to be found.

In some cases it is possible to prove directly from first principles that a problem at hand is NP-complete, but this is usually quite difficult. Most proofs proceed by constructing a polynomial-time transformation of another problem, already known to be NP-complete, into an instance of the problem at hand. It follows that, if the problem at hand could be solved in polynomial time, so could the other problem, and therefore by extension all of the problems in NP. Consequently, the problem at hand is NP-complete.

For many NP-complete decision problems there is an associated optimization problem, for which the task is to produce an optimal solution. For example, the optimization problem associated with the threading decision problem stated above is, "Find the threading of this sequence into this structure under this score function having the optimal (minimum) score." It is easy to see that the optimization problem cannot be easier than the decision problem. This is because a polynomial time solution to the optimization problem could be transformed into a polynomial time solution to the decision problem. Thus, if the optimization problem is solvable in polynomial time, then so is the decision problem. For example, if we could find the optimal threading in polynomial time, then it would be easy to compute its score (also in polynomial time). This would let us answer the decision question of whether there exists a threading with score less than K, by checking to see if the optimal score was less than K, with only one additional step. Search problems bearing this relationship to an NP-complete decision problem are called NP-hard.

Finally, we note that problems can be NP-complete for different reasons. In some cases, a polynomial time solution fails only because the numbers associated with the problem can become exponentially large in magnitude (for example, perhaps the binary bits specifying an integer are used to encode some other non-numeric information). In most cases these numbers are integers; more complicated numbers are theoretically treated as composites of several distinct integers. If a problem is NP-complete, and remains NP-complete when restricted to problem instances for which the magnitude of the largest integer is bounded by a polynomial in the problem instance size, then the problem is called NP-complete in the strong sense.

7.2 Alignment — informal sketch of proof

The protein threading problem consists of a sequence, a core, and an objective function. The decision problem is whether there exists a threading of the sequence into the core with a score under the objective function of some specified constant K or less. Call this problem "PRO-THREAD." The associated optimization problem is to produce a threading whose score is the global optimum. The bulk of the proof consists of constructing an encoding from a known NP-complete problem into PRO-THREAD. The problem we choose for this is ONE-IN-THREE 3SAT, a variant of SATISFIABILITY. The remainder of this section briefly and informally sketches the encoding. Formal details are in reference [61].

The canonical (and first) NP-complete problem is SATISFIABILITY. A problem instance consists of a set of Boolean variables, plus a set of Boolean clauses (a clause is a disjunction, or logical OR, of a set of literals; a literal is either one of the variables or the negation of one of the variables). The question is whether any setting (truth-value assignment) of the variables makes all of the clauses true simultaneously. 3SAT is a well-known variant which restricts the clauses to contain exactly three literals. ONE-IN-THREE 3SAT is a further variant of 3SAT which requires that each of the clauses be made true by *exactly one* of the three literals. All these problems are known to be NP-complete [91].

The proof that PRO-THREAD is NP-complete proceeds by showing that we can encode any arbitrary instance of ONE-IN-THREE 3SAT (does there exist a setting of the Boolean variables making all the clauses simultaneously true by exactly one literal?) as an equivalent instance of PRO-THREAD. Threadings with a score of zero encode solutions of the original ONE-IN-THREE 3SAT problem; threadings with positive scores encode failures. The equivalent encoded PRO-THREAD question is: does there exist a threading with a score of zero or less? The answer to this question is "YES" exactly when a solution exists for the original ONE-IN-THREE 3SAT problem.

The essence of the proof is this:

- Amino acids from the sequence can encode whether a Boolean variable is TRUE (by T, a threonine residue) or FALSE (by F, phenalanine); and also which literal makes a Boolean clause true (P, proline, encodes the first; Q, glutamine, the second; and R, arginine, the third, literal). In the encoded problem, the sequence **a** to be threaded is

  $$\mathbf{a} = PQRPQRPQRPQR \ldots PQRTF \ldots TFTFTFTF$$

  where we allot one "PQR" for each clause, and one "TF" for each Boolean variable.

- By making each core segment exactly one element long, it is threaded to exactly one amino acid. Consequently, any given threading assigns every core segment to one of

$\{P, Q, R, T, F\}$. (As discussed below, extensions that add "GAP" to this list are also NP-complete.)

- We can use one core segment to encode each Boolean clause, and choose which literal makes it true by threading it to P (= the first literal), Q (= the second), or R (= the third) in the sequence **a**. Similarly, one core segment encoding each Boolean variable is threaded to T (= TRUE) or F (= FALSE), and thereby chooses truth values.

- Pairs of core elements are taken as neighbors in the core (and recorded as such in the adjacency graph) exactly when the clause encoded by the first element contains a literal naming the variable encoded by the second. The edge environment label assigned is an ordered pair, $d = (i, j)$, that encodes which literal ($i = 1$, 2, or 3) is involved and whether the variable is negated ($j = Yes$ or No).

- An edge score function can be written that is zero when the edge label d is consistent with the literal choice encoded by amino acid a (as P, Q, or R) and the truth-value encoded by amino acid b (as F or T); and is one otherwise.

- By summing the edge score function over all edges, a threading score function can be written that is zero when a candidate threading encodes a truth-value and literal assignment correctly solving the original problem, and positive otherwise. The question "Does there exist a threading with a score of zero or less?" is now equivalent to the original ONE-IN-THREE 3SAT question.

- Thus, if we could solve the general PRO-THREAD problem in polynomial time, we could solve ONE-IN-THREE 3SAT in polynomial time. PRO-THREAD is NP-complete.

In fact, PRO-THREAD is NP-complete in the strong sense (i.e., is not a number problem), because the only numbers used in the construction are zero and one. The optimization problem, to produce an optimal threading, is NP-hard.

The basic proof can be used to prove that many threading methodology extensions and generalizations are also NP-hard. The general strategy in such cases is first to show that the extended problem remains in NP (because a putative solution can be checked in polynomial time), then to show that the problem has not been made easier (by exhibiting some setting of the extended parameters for which the extended problem can be made to solve PRO-THREAD). Consequently, a polynomial time solution to the extended problem would imply a polynomial time solution to the simpler PRO-THREAD. Without producing formal proofs, we sketch this for three cases of interest: allowing a core element to be unoccupied (threaded to a gap), as some dynamic programming methods permit; the inclusion of triplet or higher-order terms; and the presence of constraint equations on environment labels. Suppose we allow unoccupied elements. A method for solving this problem can be made to solve PRO-THREAD by using a score function that assigns any such threading a positive score. Similarly, extensions including triplet or higher-order terms can be made equivalent to PRO-THREAD by employing a score function that assigns all such terms a score of zero. Extensions which admit constraint equations on environment labels can be made to solve PRO-THREAD by adding tautologically true constraint equations to the original PRO-THREAD problem. Generally speaking, any related problem that includes PRO-THREAD as a special case remains NP-hard.

7.3 Selection tasks and Bayes' constants

It is easy to modify the basic proof to show that computing any of the Bayes' selection task global sums is NP-hard if pair interactions are allowed.

7.3.1 Complexity of $\mu_{\langle \mathbf{a},n,C\rangle}$

Combine the edge score functions of a given threading using multiplication instead of addition, and change the edge score function so that threadings that score zero encode failures to the original ONE-IN-THREE-3SAT problem and threadings that score 1 encode solutions. Then $\mu_{\langle \mathbf{a},n,C\rangle}$ is greater than zero exactly when a solution exists to the original ONE-IN-THREE-3SAT problem.

7.3.2 Complexity of $Z_{\langle n,C,\mathbf{t}\rangle}$

Use the same embedding as above. Again, a score of zero corresponds to a failure, and a score of 1 corresponds to a solution, of the original ONE-IN-THREE-3SAT problem. Shorten the sequence so that there is exactly one amino acid per core segment in the encoded problem, hence exactly one threading in the solution search space. Then $Z_{\langle n,C,\mathbf{t}\rangle}$ is greater than zero exactly when a solution exists to the original ONE-IN-THREE-3SAT problem.

7.3.3 Complexity of $\mu_{\langle \mathbf{a},n,C,i,t_i\rangle}$ and $\mu_{\langle \mathbf{a},n,C,I,T\rangle}$

Because $\mu_{\langle \mathbf{a},n,C\rangle} = \sum_{t_i=1}^{\tilde{n}} \mu_{\langle \mathbf{a},n,C,i,t_i\rangle}$, a polynomial-time computation for $\mu_{\langle \mathbf{a},n,C,i,t_i\rangle}$ would imply a polynomial-time computation for $\mu_{\langle \mathbf{a},n,C\rangle}$, which is NP-complete by section 7.3.1. In turn, $\mu_{\langle \mathbf{a},n,C,i,t_i\rangle}$ is a special case of $\mu_{\langle \mathbf{a},n,C,I,T\rangle}$, which therefore cannot be easier.

8 Search algorithm

The search algorithm requires as input the sequence, core template, and score function. In any threading trial, the input sequence, core template, and score function exactly define an abstract mathematical space. Each point in this search space corresponds one-to-one with a distinct alignment between the sequence and the structure. The score function assigns a scalar value (a score or pseudo energy) to each point. The global minimum score on the resulting pseudo energy landscape is the lowest score achieved by any point in the space. The global optimum alignment(s) is exactly the point(s) that achieves the global minimum score. These are well-defined objects of independent mathematical interest. They are fixed, in an exact mathematical sense, once the input sequence, core template, and score function are known

Although thereby determined, the landscape features generally are unknown. The sole task performed by the search algorithm is to report the value of the global minimum score, and to identify the global optimum threading(s) that instantiate(s) it. In contrast to approximate search methods, the branch and bound search algorithm here either finds the mathematically exact answer or it first exhausts time or space resources. The version discussed here never returns an approximate or inexact result, although fast approximate versions are possible. Of course, even for the same sequence and core template, different input score functions will produce different landscapes and different global minima. Different scoring landscapes affect the time required by such an algorithm (and hence, whether or not it converges within a specific time limit), but they cannot change the fact that the algorithm here always either finds the mathematically exact global minimum or fails to converge. The particular values of the global minimum and the best alignment, therefore, are a function only of the input; while our ability to identify them is a function of the search algorithm. Consequently, for an exact search algorithm, any agreement — or lack of it — between optimal and native alignments is a property only of the input, and not of the search algorithm.

8.1 Branch and bound algorithm

Branch and bound search [94, 95] is a computational method of finding the mathematically exact global optimum in large complex search spaces. In the best case it exploits constraints from the problem to prune the search space, so that most potential solutions are never actually examined. In the worst case it performs no better than exhaustive search. Here it is used to find the global optimum threading when both variable-length gaps and pairwise interactions are allowed [1]. Given a fixed core template, sequence, and score function, the branch and bound search algorithm here is guaranteed to find the optimal threading first, and thereafter to enumerate successive candidate threadings in score order. It provides a mathematically exact implementation for the gapped block alignment threading methodology.

The search begins with a single set containing all legal threadings. At each step, the algorithm chooses the set with the currently lowest lower bound and splits it into several subsets. The entire search space always is represented explicitly as the union of the sets created so far. After some finite number of steps, the chosen set will contain only one threading. Its score equals its lower bound. Every other set had an equal or greater lower bound, and so every other threading must have an equal or greater score. Consequently, this is the desired global optimum threading.

A set is pruned whenever its lower bound is above the global minimum score, because the global optimum threading will be discovered before that set is ever considered again. The global minimum score is unknown until the search terminates, and so pruning is implicit. If the search space may be pruned rapidly, then the search may be relatively short. Cases of multiple threadings with the same optimal score occur very rarely, and are detected automatically by continuing the search.

8.2 Algorithm pseudo-code

INITIALIZATION:

1. Compute a lower bound for the set of all threadings.
2. Initialize a sorted list to contain one entry, the set of all threadings with its lower bound.

ITERATION:

1. Remove from the list the set having the lowest lower bound.
2. If the set contains only one threading, stop and announce success. This is a global optimum threading. The procedure later may be continued from this point to enumerate successive candidate threadings in score order.
3. Otherwise, split the set into smaller subsets.
4. Compute a lower bound for each new subset.
5. Merge the new subsets into the list, sorted by lower bound.

The sorted list is implemented as a priority queue, or heap [96], for rapid access to the currently lowest lower bound.

8.3 Lower bound implementation

Efficient calculation of a strong lower bound is the essence of the branch and bound algorithm. The first part of this section describes an efficient implementation strategy. The second part describes a practical caching scheme that avoids much of the computation.

8.3.1 Implementation

This subsection describes an implementation of the lower bound $\mathrm{lb}(\mathcal{T})$ on the possible scores achieved by threadings within a set $\mathcal{T}$. As in equation 86, say that a search space axis i (i.e., the placement of core segment C_i in the sequence) is "active" in $\mathcal{T}$ if $b_i < d_i$ (i.e., the placement of core segment C_i in the sequence may vary within $\mathcal{T}$), and "inactive" if $b_i = d_i$ (C_i is fixed in $\mathcal{T}$). Note that this does not refer to pairwise or singleton contributions; both active and inactive segments may have contributions from both pairwise and singleton sources.

Separate the lower bound $\mathrm{lb}(\mathcal{T})$ into an inactive part $q(\mathcal{T})$ and an active part $r(\mathcal{T})$. These satisfy $\mathrm{lb}(\mathcal{T}) = q(\mathcal{T}) + r(\mathcal{T})$. The inactive part $q(\mathcal{T})$ sums the contributions that can be determined by knowing the exact placement of the inactive axes. These are the singleton contributions from each inactive axis, plus the pairwise contributions from each pair of inactive axes. For each subset created during the search, $q(\mathcal{T})$ is stored with the m-vectors $\mathbf{b}$ and $\mathbf{d}$ and updated at each split. The active part $r(\mathcal{T})$ estimates a lower bound on the contribution from the active axes plus their pairwise interactions with the inactive axes. It is recomputed each time the lower bound computation is done.

Use $\alpha(i)$ to indicate whether axis i is active (equation 86), and $\beta(i,j)$ to indicate whether either of axes i or j are active. Let

$$\beta(i,j) = \begin{cases} 1, & \text{if either axis } i \text{ or } j \text{ is active;} \\ 0, & \text{otherwise.} \end{cases} \tag{107}$$

β is related to α by

$$\begin{aligned}\beta(i,j) &= \alpha(i)+\alpha(j)-\alpha(i)\alpha(j) && (108)\\ &= \alpha(i)\Big(1-\alpha(j)/2\Big)+\alpha(j)\Big(1-\alpha(i)/2\Big) && (109)\end{aligned}$$

Then define

$$\begin{aligned}q(\mathcal{T}) &= \sum_i \Big(1-\alpha(i)\Big)\Big[g_1(i,b_i)+\sum_{j>i}\Big(1-\beta(i,j)\Big)g_2(i,j,b_i,b_j)\Big] && (110)\\ r(\mathcal{T}) &= \min_{\mathbf{t}\in\mathcal{T}}\sum_i \Big[\alpha(i)g_1(i,t_i)+\beta(i-1,i)g_2(i-1,i,t_{i-1},t_i)\\ &\qquad + \alpha(i)\min_{\substack{\mathbf{u}\in\mathcal{T}\\ l_j^{max}=+\infty}}\sum_{|j-i|>1}\Big(1-\alpha(j)/2\Big)g_2(i,j,t_i,u_j)\Big] && (111)\end{aligned}$$

Note that in the inner $\min_{\mathbf{u}\in\mathcal{T}}$, the ordering constraints imply that $j<i \Rightarrow u_j \leq t_i$ and $j>i \Rightarrow u_j \geq t_i$, as otherwise $g_2(i,j,t_i,u_j)=+\infty$. By convention, $g_2(j,i,t_j,t_i)=g_2(i,j,t_i,t_j)$.

Recall that $\mathrm{lb}(\mathcal{T}) = q(\mathcal{T}) + r(\mathcal{T})$. The terms in equations 110 and 111 have the same meanings as in equation 84. The inactive part $q(\mathcal{T})$ is easy to update after each split simply by accounting for newly inactive axes. The remainder of this section describes a recursive formulation of $r(\mathcal{T})$ which leads to an efficient implementation.

Define K as

$$\begin{aligned}K(i,t_i) &= \alpha(i)\Big[g_1(i,t_i)+K^*(i,m,t_i,d_m)\Big]\\ &\quad + \min_{\substack{x\geq\max(b_{i-1},t_i-\tilde{l}_{i-1})\\ x\leq\min(d_{i-1},t_i)}}\Big(K(i-1,x)+\beta(i-1,i)g_2(i-1,i,x,t_i)\Big) && (112)\end{aligned}$$

where $g_1(i,t_i)$ accounts for singleton terms, $g_2(i-1,i,x,t_i)$ forces pairwise terms containing loop scores to be consistent between $i-1$ and i, and $K^*(i,m,t_i,d_m)$ bounds the contribution from non-loop pairwise terms at $\langle i,t_i\rangle$. K^* is defined as

$$K^*(i,k,t_i,x) = \begin{cases}\min\Big(\Big[K^*(i,k-1,t_i,x)+\Big(1-\alpha(k)/2\Big)g_2(i,k,t_i,x)\Big], & \\ \qquad K^*(i,k,t_i,x-1)\Big), & \text{if } k<i-1 \text{ or } k>i+1;\\ K^*(i,k-1,t_i,x), & \text{if } i-1\leq k\leq i+1;\\ +\infty, & \text{if } x<b_k,\ x>d_k,\ k<i \text{ and } x>t_i,\\ & \quad \text{or } k>i \text{ and } x<t_i;\\ 0, & \text{otherwise.}\end{cases} \qquad (113)$$

Equation 113 treats i and t_i as parameters, and uses the assumption that $l_j^{max}=+\infty$. From equation 113 it follows that

$$K^*(i,m,t_i,d_m) = \min_{\substack{\mathbf{u}\in\mathcal{T}\\ l_j^{max}=+\infty}}\sum_{|j-i|>1}\Big(1-\alpha(j)/2\Big)g_2(i,j,t_i,u_j) \qquad (114)$$

and consequently

$$r(\mathcal{T}) = \min_x K(m, x) \tag{115}$$

as desired.

One important aspect of this lower bound computation is that the lower bound actually is instantiated on a specific threading $\mathbf{t}^{lb}$ in the outermost $\min_{\mathbf{t}\in\mathcal{T}}$ of equation 111. By keeping track of the indices x at which the minimum was actually achieved in equation 112, it is possible to follow the backtrace from the x minimizing equation 115 in order to produce $\mathbf{t}^{lb}$. This plays an important role in choosing the next split point.

A reasonably efficient implementation results from holding K and K^* in arrays and iteratively computing the array values using dynamic programming techniques. The formal computational complexity of the lower bound computation is $\mathcal{O}(m^2\tilde{n}^2)$, but this can be reduced as described next. An open problem is to devise a clever tree-structured embedding that avoids brute-force iteration, much as binary trees avoid brute-force iteration when finding the minimum value on an interval [87]. A second open problem is to strengthen the current lower bound. A third is to generalize it to higher-order core segment interactions.

8.3.2 Computing the lower bound efficiently

Most of the time is expended while computing K^* for use in computing K. However, most values of K are so bad that we actually don't need the strong bound given by K^*. In most cases, we can substitute

$$J^*(i, t_i) = \sum_{|j-i|>1} \Big(1 - \alpha(j)/2\Big) \min_{1\le x\le \tilde{n}} g_2(i, j, t_i, x) \tag{116}$$

The fact that $J^*(i,t_i) \le K^*(i, m, t_i, d_m)$ guarantees that the result is a valid lower bound. Computing $J^*(i,t_i)$ is very fast because $\min_x g_2(i,j,t_i,x)$ can be precomputed and stored for each (i,j,t_i), and the computation then reduces to sums of a few array references.

In fact, it is sufficient if we ensure that $\mathbf{t}^{lb}$ and the value of its associated lower bound are computed using K^*; all other cases may use J^*. To do this, we record all indices $\langle i, t_i\rangle$ that have ever appeared in $\mathbf{t}^{lb}$ during any lower bound computation. Equation 112 is computed using K^* (equation 113) for each such $\langle i, t_i\rangle$, and using J^* (equation 116) otherwise. Specifically, let

$$\gamma(i, t_i) = \begin{cases} 1, & \text{if } \langle i, t_i\rangle \text{ ever appeared in any } \mathbf{t}^{lb}; \\ 0, & \text{otherwise.} \end{cases} \tag{117}$$

$$\begin{aligned} K^{\text{fast}}(i, t_i) = {} & \alpha(i)\Big[g_1(i,t_i) + \gamma(i,t_i)K^*(i,m,t_i,d_m) + \Big(1-\gamma(i,t_i)\Big)J^*(i,t_i)\Big] \\ & + \min_{\substack{x \ge \max(b_{i-1}, t_i - \tilde{l}_{i-1}) \\ x \le \min(d_{i-1}, t_i)}} \Big(K^{\text{fast}}(i-1, x) + \beta(i-1,i)g_2(i-1,i,x,t_i)\Big) \end{aligned} \tag{118}$$

Equation 118 is used in place of equation 112 in order to avoid most invocations of equation 113.

It remains to ensure that the current computation did not reach a new $\langle i, t_i\rangle$ appearing in the current $\mathbf{t}^{lb}$ for the first time, by checking $\gamma(i, t_i^{lb})$ for each $\langle i, t_i^{lb}\rangle$. If $\gamma(i,t_i^{lb}) = 0$ for any i, then that $\gamma(i, t_i^{lb})$ must be set to 1 and the lower bound computation repeated. In practice, only a few such $\langle i, t_i \rangle$ ever appear. Because most values of K are sufficiently bad, the difference between K^* and J^* doesn't matter in most cases. Cases where it does matter typically are identified early on, and subsequently very little repeat computation is done.

An efficient implementation might scale and round the input in order to use fast integer arithmetic; keep arrays as nested pointers in order to avoid multi-dimensional array references; lay out large arrays in small pieces in order to minimize disk paging; precompute or cache values where possible; and so on. A parallel MIMD implementation could distribute subsets among arbitrarily many processors. A parallel SIMD implementation could embed the array computations in a connected grid of processors.

9 Computational Experiments

This section presents the branch and bound search algorithm's computational behavior and current limits. It shows that the search can succeed in many practical cases, and illustrates the relationship between problem size and computational resources required. Detailed computational analyses are based on the score function of Bryant & Lawrence [27], because it has the highest convergence rate found (99.8%) and thus gives a picture of performance spanning thirty orders of magnitude in search space size ($< 10^1$ to $> 10^{31}$). Five score functions [27, 28, 29, 55, 58] are used to illustrate general trends. Every example described has been run under all five score functions employed, and yields the same qualitative behavior (often with substantial variation in detail).

Two of the five score functions shown below [27, 58] directly provide loop (or loop reference state) score terms as part of their score function. The other three [28, 29, 55] here require an auxiliary loop score function. This was set to zero for the timing analysis, which therefore depends only on previously published values or theories. For biological examples [1] we set it proportional to a simple log odds ratio, $\log(P(a|\text{loop})/P(a))$, summed over all amino acids a in the loop. Here $P(a)$ is the prior probability of a and $P(a|\text{loop})$ is the probability of observing a in a loop region.

9.1 Core template library

We developed a library of core templates taken from 58 non-homologous, monomeric, single-domain, soluble, globular proteins representing diverse structure types (described in table 2). We believe this to be one of the simplest interesting test cases: statistical artifacts arising from much smaller test sets are avoided, and the proteins require no arbitrary decisions about hydrophobic face packing on domain or multimer boundaries. In order to avoid any subjective bias in core definition, core segments were exactly the main-chain plus β-carbon atoms (inferred for glycine) of α-helices and β-strands taken from the Brookhaven Protein Data Bank feature tables [97], or if not present were computed from atomic coordinates using DSSP [98] (smoothed as in Stultz *et al.* [59]). All side-chains were replaced by alanine, in order to assign core template environments independent of the original amino acid identities. Loops were discarded. The resulting core templates were equivalent to a backbone trace plus β-carbons of the core secondary structure, annotated as required by the score function. An even more abstract template would use ideal coordinates. We sought to reduce residual traces of the structure's original primary sequence (sequence memory) and loop lengths (gap memory), as otherwise threading alignment accuracy on distant structural homologs may suffer (see discussions by references [40, 41, 99]).

We exhaustively threaded every library sequence through every library core template. This created 3,364 sequence-template pairs, each consisting of a single fixed sequence and core template. Template loops assigned length zero or one by the crystallographer were treated as fixed-length because they usually reflect constrained "kinks" in the secondary structure. In all other cases we considered all physically realizable loop lengths that maintained core segment topological order. Any loop length that could be proven to break the main-chain or violate excluded atomic volumes was discarded as illegal. Consequently, 833 sequence-template pairs were discarded *a priori* because the sequence was too short to occupy the template under any legal loop assignment. With the remaining 2,531 admissable pairs we searched for the global optimum threading under all five score functions considered. This resulted in a total of 12,655 legal trials, where each trial corresponded to a search for the global optimum threading given a fixed sequence, core template, and score function. Trials were run on a desktop workstation DEC Alpha 3000-M8000, using public-domain CMU Common Lisp [100], and were terminated at the computational limit of two hours. For each trial, we computed the size of the search space of legal threadings and recorded the elapsed time required to find its global optimum threading.

9.2 Problem size and computation time

In a total of 12,109 trials (96%) the search converged within two hours; in 488 trials (4%) time was exhausted first; and in 58 trials (0.5%) space was exhausted first. Figure 6 shows the time required to find the global optimum in every convergent trial under all five score functions, as a function of search space size. On a DEC Alpha 3000-M8000 desktop workstation running LISP, we have identified the global optimum threading in NP-hard search spaces as large as 9.6×10^{31} at rates ranging as high as 6.8×10^{28} equivalent threadings per second, most of which were pruned before they were ever explicitly examined.

Figure 6 about here.

Table 2 shows detailed timing results for self-threading each sequence into its own core template using the Bryant & Lawrence (1993) score function [27], abbreviated "BL93." Protein size is stated in terms of sequence length and number of core segments; search space growth is exponential in number of core segments, but in practice proteins are roughly one-half secondary structure and so the two measures are roughly proportional. Total elapsed time is resolved into initialization and search components, showing that the fast search does not require a prohibitively long initialization. The data in table 2 appear together with all non-self-threading trials in figure 6 in the plot labeled "BL93," transformed by $x = \log_{10}$("Search Space Size") and $y = \log_{10}$("Total Seconds").

Table 2 about here.

Table 3 shows the fraction of trials that converged in each case, the total and per-trial time required, and the log-log regression slopes and intercepts, across all five score functions used. It compares native and non-native threadings for each graph in figure 6, and gives the pooled results of all trials. Table 3 summarizes table 2 in the row labeled "BL93 Native."

Table 3 about here.

Figure 7 shows histograms of number of trials and total time expended finding optimal threadings under Bryant & Lawrence [27], grouped by search space size. In 81% of all trials, the search space contained fewer than 10^{15} legal threadings. However, the searches in those same trials expended only 11% of the total time. Conversely, only 4% of all trials involved a search space that contained more than 10^{20} threadings, but their searches expended 71% of the total time. Figure 7 corresponds to figure 6 in the graph labeled "BL93."

Figure 7 about here.

---

10 Discussion

It is clear from table 2 that the search algorithm is successful at drastically reducing the portion of search space that actually need be examined. This allows the algorithm to find the optimal threading in vastly larger search spaces than heretofore possible. Figure 6 shows that this behavior is characteristic of a wide variety of score functions. Larger search spaces do require more time, as expected, but in most cases examined an exact search could be accomplished within reasonable limits.

Because the algorithm search behavior depends on the input score function, sequence, and core template, it is difficult to give an analytic statement of its average-case time complexity. Figure 6 shows that the log-log relationships remain approximately linear over nearly twenty-five orders of magnitude. For the range of data considered, the regression slopes in table 3 were between 0.12 and 0.24 and the intercepts were between -1.16 and -0.39. Changes in raw speed due to different computer languages or hardware should affect the intercept but leave the slope unchanged, producing a constant vertical offset in figure 6. Because $\log y = a \log x + b$ implies $y = e^b x^a$, the search time in seconds was approximately proportional to between the fourth and eighth root of the search space size and the proportionality constant was between 0.3 and 0.7. Differences in the underlying search space landscapes probably give rise to the considerable scatter about the central tendency and the variation in timing behavior between score functions. We expect that further speed increases can be achieved by parallelizing the algorithm. It has efficient implementation strategies for both Single Instruction Multiple Data (SIMD) and Multiple Instruction Multiple Data (MIMD) parallel computers. Tighter lower bound formulae (cf. equation 84) would decrease the slope in figure 6, which would lead to greater leverage in larger search spaces.

Due to the exponential nature of the search, most of the time is expended during the few trials that search the very largest search spaces. However, most trials involve search spaces that are substantially smaller, hence more quickly searched. As figure 7 shows, most results can be obtained for relatively little computational effort. All five score functions converged quickly on almost all trials that involved a small to medium sized search space, e.g. of size 10^{20} or less. An important implication is that core templates at the level of super-secondary structure or domain motifs should thread very quickly. This result should greatly encourage efforts to recognize protein structure building blocks and assemble them hierarchically.

10.1 Scoring schemes

Our recent experiences and those of others [1, 36] on current threading structural environment descriptors and scoring potential functions strongly suggest a need for better protein structural environment definitions and objective functions. There are at least two potential problems in all of the current scoring schemas. The first is a standard statistical problem associated with limited data or low counts. Due in part to the limited size of the current database, some amino acid pairs are sparsely populated in particular structural environments. For example, if there are four or more structural environments associated with each modeled fold position, then there are nearly four hundred times sixteen different independent pairwise score terms to estimate. However, some amino acids are quite rare, and currently there are only a hundred or so completely independent single fold protein structures determined [85, 101, 102, 103, 104]. Thus, for many neighboring pairs in particular environments there are very few or zero observations. In such cases the threading score functions are sensitive to fluctuations in the observed number of occurrences. For example, methods that use the logarithms of probabilities or derived odds ratios are sensitive to small frequencies because the logarithm becomes numerically unstable as its argument approaches zero. This rare observation problem compounds the testing or validation of any threading score function because score functions may tend to "memorize" the particular proteins in the data set used to derive score function parameters. This is, of course, due to the fact that a rare event can provide sequence-to-structure memory in the score function, thus invalidating the test. Cross-validated tests (jack-knife, hold-one-out, etc.) are critical (see the discussion by Ouzounis *et al.* [40]).

The second potential problem (and likely the more limiting) is in the definition of neighbor and/or pairwise contact environments. The question appears to be, when are two amino acid positions meaningful neighbors? If they are physically close? If they have a particular geometric relationship to one another? If their side chain atoms interact energetically? The simplest side chain independent definition of neighboring structural positions is one that depends only on the physical distance between the alpha or beta carbon pairs. Thus in a number of scoring schemes, the pairwise preferences of amino acids are obtained by counting the occurrence frequencies at which each type of amino acid's beta carbon is within a particular distance of another type of amino acid's beta carbon. This simple definition is not obviously related to the physical pairwise interactions expected to affect protein folding. Note, two amino acids on opposite sides of a beta sheet might satisfy the distance conditions, but fail to make any physical contact. More to the point is the fact that two close amino acids, even on the same side of a beta sheet or alpha helix, may make little or no physical energetic contact due to the positioning of their side chain rotamers. The rotamer of an amino acid defines the direction in which the rest of its side chain atoms point. Most amino acids have beta carbons with four atoms bonded in a tetrahedral configuration, with freedom to rotate about the alpha-beta axis when in solution. Two very close amino acids whose side chains point away from each other normally will have no significant contact. Thus any simple spatial definition of neighborness used in calculating a threading scoring scheme directly from the occupation frequencies of nearby positions will be a mix of noninteracting "chance" neighbors and truly interacting neighbors. That would give a potential noise to signal ratio of at least two to one.

11 Conclusions

One of the most probable limiting factors of the threading predictability appears to be current score functions. There are obviously strong correlations between what particular type of amino acid is found in a given structural position and who its neighbors are. However, one needs to distinguish between the fact that a buried amino acid is most likely to be strongly hydrophobic and the fact that most of its neighbors also being buried will be hydrophobic — that is, amino acids sharing similar local environments will have correlated properties, but not necessarily because they interact. Nearly all of the current threading scoring functions contain both local and nonlocal terms, the former being the local environment as a function of the position only within the overall structure, and the latter being a function of what other amino acids are brought into the neighborhood by their alignment with the rest of the structure. Currently, in most cases, under current score functions a small fraction of the non-native threadings in the search space continue to score better than the native. Because the search space may be combinatorically large, however, this small fraction may include very many individual threadings.

Why try to continue to work on a method that seems to have so many problems? For one thing, it may work, even with the current limitations. In any particular test case the pairwise noise may happen to "average out" and the local environmental preferences alone may provide good predictions; in fact, this often happens in practice. In addition, there are many extensions to the threading approach that should prove very useful. By combining functional diagnostic patterns of amino acids with threading models, one should be able to extract information as to the most likely positions of those key residues. By extending the threading concept to the threading or alignment of amino acid sequences with partially determined X-ray electron densities [2] or NMR data, one should be able to speed up structure determinations. By using threading as a design tool, one should be able to engineer sequences for particular folds.

Acknowledgments

Ilya Muchnik helped develop the probabilistic framework within which this work is situated. We thank Melissa Cline, Lisa Tucker-Kellogg, Loredana Lo Conte, Sophia Zarakhovich and Srikar Rao for their work on core modelling and score functions; Gene Myers and Jim Knight for discussions of the mathematical formalism; Tomás Lozano-Perez and Patrick Winston for discussions of computational protein folding; Janice Glasgow, David Haussler and Alan Lapedes for applications and extensions; and Steve Bryant, Gordon Crippen, Chip Lawrence, Vladimir Maiorov, and Manfred Sippl for discussions of their score functions. Comments from Nick Steffen improved the presentation. Special thanks to all crystallographers who deposited their coordinates in the international scientific databases.

This chapter describes research performed at the Department of Information and Computer Science of the University of California, Irvine; the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology; and the BioMolecular Engineering Research Center of Boston University. Support for the first author is provided in part by a CAREER grant from the National Science Foundation. Support for the Artificial Intelligence Laboratory's research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-91-J-4038. Support for the BioMolecular Engineering Research Center is provided in part by the National Library of Medicine under grant number P41 LM05205-14, and by the National Science Foundation under grant number BIR-9121546. The contents of this chapter are solely the responsibility of the authors and do not necessarily represent the official views of the granting agencies.

References

[1] R. H. Lathrop and T. F. Smith (1996) *J. Mol. Biol.* 255, 641–665.

[2] K. Baxter, E. Steeg, R. H. Lathrop, J. Glasgow, and S. Fortier (1996) From electron density and sequence to structure: Integrating protein image analysis and threading for structure determination In D. J. States, P. Agarwal, T. Gaasterland, L. Hunter, and L. Smith (Eds.), *Proc. Intl. Conf. on Intelligent Systems and Molecular Biology* pp. 25–33 Menlo Park, California. AAAI Press.

[3] T. F. Smith, L. Lo Conte, J. Bienkowska, B. Rogers, C. Gaitatzes, and R. H. Lathrop (1997) The threading approach to the inverse folding problem In S. Istrail, R. Karp, T. Lengauer, P. Pevzner, R. Shamir, and M. Waterman (Eds.), *Proc. Intl. Conf. on Computational Molecular Biology* pp. 287–292 New York. ACM Press.

[4] T. F. Smith, L. Lo Conte, J. Bienkowska, C. Gaitatzes, R. G. Rogers Jr., and R. H. Lathrop (1997) *J. Comp. Biol.* 4, 217–225.

[5] T. F. Smith, R. H. Lathrop, and F. E. Cohen (1996) The identification of protein functional patterns In J. Collado-Vides, B. Magasanik, and T. F. Smith (Eds.), *Integrative Approaches to Molecular Biology* pp. 29–61 Cambridge, Massachusetts. MIT Press.

[6] G. Kolata (1986) *Science* 233, 1037–1039.

[7] C. Herzfeld (chair) et al. (1990) Grand challenges: High performance computing and communications. Technical report Report by the Committee on Physical, Mathematical, and Engineering Sciences of the U.S. Office of Science and Technology Policy.

[8] S. J. Weiner, P. A. Kollman, D. A. Case, U. C. Singh, C. Ghio, G. Alagona, S. Profeta, and P. Weiner (1984) *J. Am. Chem. Soc.* 106, 765–784.

[9] C. L. Brooks, M. Karplus, and B. M. Pettitt (1990) *Proteins: A theoretical perspective of dynamics, structure, and thermodynamics* John Wiley and Sons New York.

[10] T. E. Creighton (1983) *Biopolymers* 22, 49.

[11] J. Novotný, A. A. Rashin, and R. E. Bruccoleri (1988) *Proteins: Structure, Function, and Genetics* 4, 19–30.

[12] J. Moult, J. T. Pedersen, R. Judson, and K. Fidelis (1995) *Proteins: Structure, Function, and Genetics* 23, ii–iv.

[13] R. Srinivasan and G. D. Rose (1995) *Proteins: Structure, Function, and Genetics* 22, 81–99.

[14] J. Skolnick, A. Kolinski, and A. R. Ortiz (1997) *J. Mol. Biol.* 265, 217–241.

[15] C. Chothia (1992) *Nature* 357, 543–544.

[16] C. A. Orengo, D. T. Jones, and J. M. Thornton (1994) *Nature* 372, 631–634.

[17] J. Greer (1990) *Proteins: Structure, Function, and Genetics* 7, 317–333.

[18] Q. Zheng, R. Rosenfeld, S. Vajda, and C. DeLisi (1993) *Protein Science* 2, 1242–1248.

[19] J. Desmet, M. De Maeyer, B. Hazes, and I. Lasters (1992) *Nature* 356, 539–542.

[20] C. Mandal and D. S. Linthicum (1993) *J. Computer-Aided Mol. Design* 7, 199–224.

[21] D. Sankof and J. B. Kruskal (Eds.) (1983) *Time warps, string edits and macromolecules* Addison-Wesley Reading, Massachusetts.

[22] E. Pennisi (1997) *Science* 277, 1432–1434.

[23] L. Holm and C. Sander (1993) *J. Mol. Biol.* 233, 123–138.

[24] L. Holm and C. Sander (1994) *Nucl. Acids Res.* 22, 3600–3609.

[25] J. S. Richardson (1981) *Adv. Protein Chem.* 34, 167–339.

[26] R. Lüthy, J. U. Bowie, and D. Eisenberg (1992) *Nature* 356, 83–85.

[27] S. H. Bryant and C. E. Lawrence (1993) *Proteins: Structure, Function, and Genetics* 16, 92–112.

[28] V. N. Maiorov and G. M. Crippen (1992) *J. Mol. Biol.* 227, 876–888.

[29] M. J. Sippl (1993) *J. Computer-Aided Mol. Design* 7, 473–501.

[30] J. Bowie and D. Eisenberg (1993) *Current Opinion in Structural Biol.* 3, 437–444.

[31] S. H. Bryant and S. F. Altschul (1995) *Current Opinion in Structural Biol.* 5, 236–244.

[32] J. S. Fetrow and S. H. Bryant (1993) *Bio/Technology* 11, 479–484.

[33] R. L. Jernigan and I. Bahar (1996) *Current Opinion in Structural Biol.* 6, 195–209.

[34] D. T. Jones and J. M. Thornton (1993) *J. Computer-Aided Mol. Design* 7, 439–456.

[35] D. T. Jones and J. M. Thornton (1996) *Current Opinion in Structural Biol.* 6, 210–216.

[36] C. M.-R. Lemer, M. J. Rooman, and S. J. Wodak (1995) *Proteins: Structure, Function, and Genetics* 23, 337–355.

[37] M. J. Sippl (1995) *Current Opinion in Structural Biol.* 5, 229–235.

[38] S. J. Wodak and M. J. Rooman (1993) *Current Opinion in Structural Biol.* 3, 247–259.

[39] G. M. Crippen (1996) *Proteins* 26, 167–71.

[40] C. Ouzounis, C. Sander, M. Scharf, and R. Schneider (1993) *J. Mol. Biol.* 232, 805–825.

[41] R. B. Russell and G. J. Barton (1994) *J. Mol. Biol.* 244, 332–350.

[42] P. D. Thomas and K. A. Dill (1996) *J. Mol. Biol.* 257, 457–469.

[43] W. of Ockham (ca. 1319) *Commentary on the Sentences* of Peter Lombard (the *Reportario*) Cited p. 35n in [105].

[44] M. J. Sippl, M. Hendlich, and P. Lackner (1992) *Protein Science* 1, 625–640.

[45] A. Kolinski, J. Skolnick, and A. Godzik (1996) An algorithm for prediction of structural elements in small proteins In L. Hunter and T. Klein (Eds.), *Proc. Pacific Symposium on Biocomputing '96* pp. 446–460 Singapore. World Scientific.

[46] K. T. Simons, C. Kooperberg, E. Huang, and D. Baker (1997) *J. Mol. Biol.* 268, 209–225.

[47] T. F. Smith (1995) *Science* 268, 958–959.

[48] R. Abagyan, D. Frishman, and P. Argos (1994) *Proteins: Structure, Function, and Genetics* 19, 132–140.

[49] A. Bauer and A. Beyer (1994) *Proteins: Structure, Function, and Genetics* 18, 254–261.

[50] F. U. Bowie, R. Lüthy, and D. Eisenberg (1991) *Science* 253, 164–170.

[51] A. Godzik, A. Kolinski, and J. Skolnick (1992) *J. Mol. Biol.* 227, 227–238.

[52] M. Hendlich, P. Lackner, S. Weitckus, H. Flöckner, R. Froschauer, K. Gottsbacher, G. Casari, and M. J. Sippl (1990) *J. Mol. Biol.* 216, 167–180.

[53] E. S. Huang, S. Subbiah, and M. Levitt (1995) *J. Mol. Biol.* 252, 709–720.

[54] J.-P. A. Kocher, M. J. Rooman, and S. J. Wodak (1994) *J. Mol. Biol.* 235, 1598–1613.

[55] S. Miyazawa and R. L. Jernigan (1985) *Macromolecules* 18, 534–552.

[56] Y. Wang, L. Lai, Y. Han, X. Xu, and Y. Tang (1995) *Proteins: Structure, Function, and Genetics* 21, 127–129.

[57] M. Wilmanns and D. Eisenberg (1993) *Proc. Natl. Acad. Sci. USA* 90, 1379–1383.

[58] J. V. White, I. Muchnik, and T. F. Smith (1994) *Mathematical Biosciences* 124, 149–179.

[59] C. M. Stultz, R. Nambudripad, R. H. Lathrop, and J. V. White (in press) Predicting protein structure with probabilistic models In N. Allewell and C. Woodward (Eds.), *Protein Folding and Stability* Greenwich. JAI Press.

[60] M. J. Sippl (1990) *J. Mol. Biol.* 213, 859–883.

[61] R. H. Lathrop (1994) *Protein Engineering* 7, 1059–1068.

[62] T. Akutsu and S. Miyano (1997) On the approximation of protein threading In S. Istrail, R. Karp, T. Lengauer, P. Pevzner, R. Shamir, and M. Waterman (Eds.), *Proc. Intl. Conf. on Computational Molecular Biology* pp. 3–8 New York. ACM Press.

[63] C. A. Orengo and W. R. Taylor (1990) *J. Theor. Biol.* 147, 517–551.

[64] W. R. Taylor and C. A. Orengo (1989) *J. Mol. Biol.* 208, 1–22.

[65] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton (1993) *Science* 262, 208–14.

[66] D. T. Jones, W. R. Taylor, and J. M. Thornton (1992) *Nature* 358, 86–89.

[67] H. Flöckner, M. Braxenthaler, P. Lackner, M. Jaritz, M. Ortner, and M. J. Sippl (1995) *Proteins: Structure, Function, and Genetics* 23, 376–386.

[68] V. N. Maiorov and G. M. Crippen (1994) *Proteins: Structure, Function, and Genetics* 20, 167–173.

[69] S. A. Benner, M. A. Cohen, and G. H. Gonnet (1993) *J. Mol. Biol.* 229, 1065–1082.

[70] A. V. Finkelstein and B. Reva (1991) *Nature* 351, 497–499.

[71] T. Madej, J.-F. Gibrat, and S. H. Bryant (1995) *Proteins: Structure, Function, and Genetics* 23, 356–369.

[72] S. C. Bagley, L. Wei, C. Cheng, and R. B. Altman (1995) Characterizing oriented protein structural sites using biochemical properties In C. Rawlings, D. Clark, R. Altman, L. Hunter, T. Lengauer, and S. Wodak (Eds.), *Proc. 3rd Intl. Conf. on Intelligent Systems for Mol. Biol.* pp. 12–20 Menlo Park, California. AAAI Press.

[73] T. Grossman, R. Farber, and A. Lapedes (1995) Neural net representations of empirical protein potentials In C. Rawlings, D. Clark, R. Altman, L. Hunter, T. Lengauer, and S. Wodak (Eds.), *Proc. 3rd Intl. Conf. on Intelligent Systems for Mol. Biol.* pp. 154–161 Menlo Park, California. AAAI Press.

[74] A. Tropsha, R. K. Singh, I. I. Vaisman, and W. Zheng (1996) Statistical geometry analysis of proteins: Implications for inverted structure prediction In L. Hunter and T. Klein (Eds.), *Proc. Pacific Symposium on Biocomputing '96* pp. 614–623 Singapore. World Scientific.

[75] P. J. Munson and R. K. Singh (1997) Multi-body interactions within the graph of protein structure In T. Gaasterland, P. Karp, K. Karplus, C. Ouzounis, C. Sander, and A. Valencia (Eds.), *Proc. 5th Intl. Conf. on Intelligent Systems for Mol. Biol.* pp. 198–201 Menlo Park, California. AAAI Press.

[76] T. Bayes (1764) *Philosophical Transactions of the Royal Society of London* 53, 370–418 Reprinted pp. 131–153 in [106].

[77] G. E. Box and G. C. Tiao (1973) *Bayesian inference in statistical analysis* Addison-Wesley Reading, Massachusetts.

[78] J. A. Hartigan (1983) *Bayes Theory* Springer-Verlag New York.

[79] G. E. Arnold, A. K. Dunker, S. J. Johns, and R. J. Douthart (1992) *Proteins: Structure, Function, and Genetics* 12, 382–399.

[80] R. L. Dunbrack, Jr. and F. E. Cohen (1997) *Protein Science* 6, 1661–1681.

[81] M. J. Thompson and R. A. Goldstein (1996) *Proteins: Structure, Function, and Genetics* 25, 38–47.

[82] L. Hunter and D. J. States (1992) *IEEE Expert* 7, 67–75.

[83] J. V. White, C. M. Stultz, and T. F. Smith (1994) *Mathematical Biosciences* 191, 35–75.

[84] A. G. Murzin, S. E. Brener, T. Hubbard, and C. Chothia (1995) *J. Mol. Biol.* 247, 536–540.

[85] L. Holm and C. Sander (1996) *Science* 273, 595–602.

[86] G. L. Steele Jr. (1990) *Common Lisp: the Language* Digital Press Bedford, Massachusetts.

[87] R. Sedgewick (1990) *Algorithms in C* Addison-Wesley Reading, Massachusetts.

[88] A. S. Fraenkel (1993) *Bull. Math. Biol.* 55, 1199–1210.

[89] J. T. Ngo and J. Marks (1992) *Protein Engineering* 5, 313–321.

[90] R. Unger and J. Moult (1993) *Bull. Math. Biol.* 55, 1183–1198.

[91] M. R. Garey and D. S. Johnson (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness* W. H. Freeman and Company New York.

[92] J. E. Hopcroft and J. D. Ullman (1979) *Introduction to Automata Theory, Languages, and Computation* Addison-Wesley Reading, Massachusetts.

[93] H. R. Lewis and C. H. Papadimitriou (1979) *Elements of the Theory of Computation* Prentice-Hall Englewood Cliffs, New Jersey.

[94] P. H. Winston (1993) *Artificial Intelligence* Addison-Wesley Reading, Massachusetts third edition.

[95] V. Kumar (1992) Search, branch-and-bound In S. C. Shapiro (Ed.), *Encyclopedia of artificial intelligence, vol. 2* pp. 1468–1472 New York. John Wiley and Sons.

[96] A. V. Aho, J. E. Hopcroft, and J. D. Ullman (1982) *Data Structures and Algorithms* Addison-Wesley Reading, Massachusetts.

[97] 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 (1977) *J. Mol. Biol.* 112, 535–542.

[98] W. Kabsch and C. Sander (1983) *Biopolymers* 22, 2577–2637.

[99] B. Rost and C. Sander (1994) *Proteins: Structure, Function, and Genetics* 20, 216–226.

[100] R. A. MacLachlan (1992) Cmu common lisp user's manual Technical report School of Computer Science, Carnegie Mellon University Pittsburgh, Pennsylvania CMU Common Lisp source code and executables are freely available via anonymous FTP from `lisp-rt1.slisp.cs.cmu.edu (128.2.217.9)` and `lisp-rt2.slisp.cs.cmu.edu (128.2.217.10)`.

[101] C. A. Orengo, A. D. Michie, S. Jones, D. T. Jones, M. B. Swindells, and J. M. Thornton (1997) *Structure* 5, 1093–1108.

[102] S. E. Brenner, C. Chothia, and T. Hubbard (1997) *Current Opinion in Structural Biol.* 7, 369–376.

[103] L. Holm and C. Sander (1997) *Nucleic Acids Res.* 25, pp 231–234.

[104] A. D. Michie, C. A. Orengo, and J. M. Thornton (1996) *J. Mol. Biol.* 262, 168–185.

[105] G. Leff (1975) *William of Ockham* Manchester University Press Manchester, UK.

[106] E. S. Pearson and M. G. Kendall (Eds.) (1970) *Studies in the History of Statistics and Probability* Charles Griffin London.

Figure Legends

Legend for figure 1.

Selecting a core from a structural library. Cores in the structural library are rank ordered by their probability according to equation 28. The globally most probable core is shown selected. It is also possible to enumerate all cores in order of probability, or to sample most-probable cores.

Legend for figure 2.

A schematic view of the gapped block alignment approach to protein threading (adapted from [1]).
(a) Conceptual drawing of two structurally similar proteins and a common core of four secondary structure segments (dark lines, I-L). Note that there is no restriction on core segment length, which may range from a single residue position upwards. To form the core templates used here, side-chains were replaced by a methyl group resulting in polyalanine, and loops or variable regions were removed resulting in discrete core segments.
(b) Abstract core template showing spatial adjacencies (interactions). Small circles represent amino acid positions (core elements), and thin lines connect neighbors that interact in the objective function. The structural environments and interacting positions will be recorded for later use by the objective function.
(c) Illustration of the combinatorically large number of threadings (sequence-structure alignments) possible with a novel sequence. t_x^a indexes the sequence amino acid placed into the first element of segment X. Sequence regions between core segments become connecting turns or loops, which are constrained to be physically realizable. All alignment gaps are confined to turn or loop regions.
(d) A sequence is threaded through the core template by placing successive sequence amino acids into adjacent core elements. Alignments are rank-ordered by their probability according to equation 24. The globally most probable alignment is shown selected. It is also possible to enumerate all alignments in order of probability, or to sample the near-optimal alignments.

Legend for figure 3.

Selecting a core and alignment jointly. Conceptually, every core template of the structural library (Str. Lib., C_i) is used to generate a pool of all possible sequence-structure alignments (All Aligns., $\cup_i \mathcal{T}[C_i, n]$) with the input sequence. The pooled (core, alignment) pairs are rank ordered by probability according to equation 34. The globally most probable core template and alignment pair is shown selected. It is also possible to enumerate all (core, alignment) pairs in order of probability, or to sample near-optimal pairs.

Legend for figure 4.

Schematic view of the general-case information transformation pathways.

Legend for figure 5.

Defining and splitting sets of threadings. Sets used in the branch-and-bound search are defined by lower and upper limits (dark arrows, labeled b_x^a and d_x^a for segment X) on the sequence amino acid placed into the first core element of each segment. The set consists of all legal threadings such that the first element of each segment X is within the interval $[b_x^a, d_x^a]$. A set is split into subsets by choosing one core segment (here, segment I) and one split point (dark interior arrow). Its interval is split into sub-intervals (1) less than, (2) equal to, and (3) greater than the split point.

Legend to figure 6 — Slow Exponential Growth.

The time required to find the global minimum is shown on log-log axes as a function of search space size. All sequences and all core templates in our library were threaded through each other under every score function considered. All physically realizable loops were considered. Occassionally the crystallographer assigns a loop length of zero or one; this usually reflects a constrained "kink" in secondary structure, not a true loop, and was left unchanged. The graph for each score function shows every trial that converged under that score function. PDB codes are shown in table 2.

Score functions: "BL93" = Bryant & Lawrence (1993) [27]; "MC92" = Maiorov & Crippen (1992) [28]; "MJ85" = Miyazawa & Jernigan (1985) [55]; "S93" = Sippl (1990, 1993) [29]; "WMS94" = White *et al.* (1994) [58].

Timing resolution is one second. The dashed line corresponds to our computational limit of two hours. All physically realizable loop lengths were admitted, but gaps provably breaking the chain or violating excluded atomic volumes were prohibited. Trials were performed using CMU Common Lisp [100] running on a DEC Alpha 3000-M8000 desktop workstation.

Legend to figure 7 — Histograms of number of trials ("N") and total time expended ("T").

Histograms of number of trials ("N", white bars, dotted line) and total time expended ("T", striped bars, solid line), grouped by search space size and expressed as fractions of the total. Trials and search space sizes reflect the 2,531 legal sequence-template pairs that result from threading every sequence through every core template in our library (PDB codes in table 2). Time expended reflects trials using the Bryant & Lawrence (1993) score function [27], as shown in figure 6, BL93. Each non-convergent trial expended two hours.

The histograms group trials according to $\log_{10}(SearchSpaceSize)$. For example, "0-5" indicates the trials such that the search space size is between 10^0 and 10^5, "N" indicates the fraction of total trials having search space sizes in that range, and "T" indicates the fraction of the total time that was expended on them.

Table captions

Caption to table 1 — Notational usage of this paper.

Notational usage of this paper.

Caption to table 2 — Detailed Timing, Bryant and Lawrence (1993) [27], DEC Alpha 3000-M8000.

Detailed data for all self-threading cases in figure 6, BL93. Experimental conditions are described in figure 6. "PDB Code" is the locus name in the Brookhaven Protein Data Bank [97]. "Search Space Size" is the size of the search space within which the algorithm finds the optimal threading. Note that the algorithm *does not actually calculate* all of these threadings, which is one of its critical strengths. "Total (Search-only) Seconds" is total (in parentheses, only the search component) real elapsed clock time. Total time includes reading the protein sequence and core template from files, all datastructure initialization, and the search itself, but not reading the score function parameter files (in a predictive setting these would be memory-resident). "Number of Search Iterations" is the number of times the loop labeled "Iteration" in section 8.2 was executed. "Equivalent Threadings per Second" and "Equivalent Threadings per Iteration" are the ratio of search space size to the respective quantities. The ratios represent, respectively, the speed required for exhaustive search to achieve the same time, and the number of threadings that exhaustive search would examine per iteration of pruned search.

Caption to table 3 — Convergence, time, slopes, and intercepts.

Experimental conditions are described in figure 6. "Native" refers to threading sequences into their native cores, "Non-Native" into non-native cores. "Hours Required" is the total time consumed by all searches. Searches that did not converge consumed two hours before being terminated. "Slope" and "Interc." are respectively the slope and intercept (a and b in $y = ax + b$) of the best fitting regression line to the graphs in figure 6, discarding points at $y = 0$. "Pooled" accumulates the results of all trials.

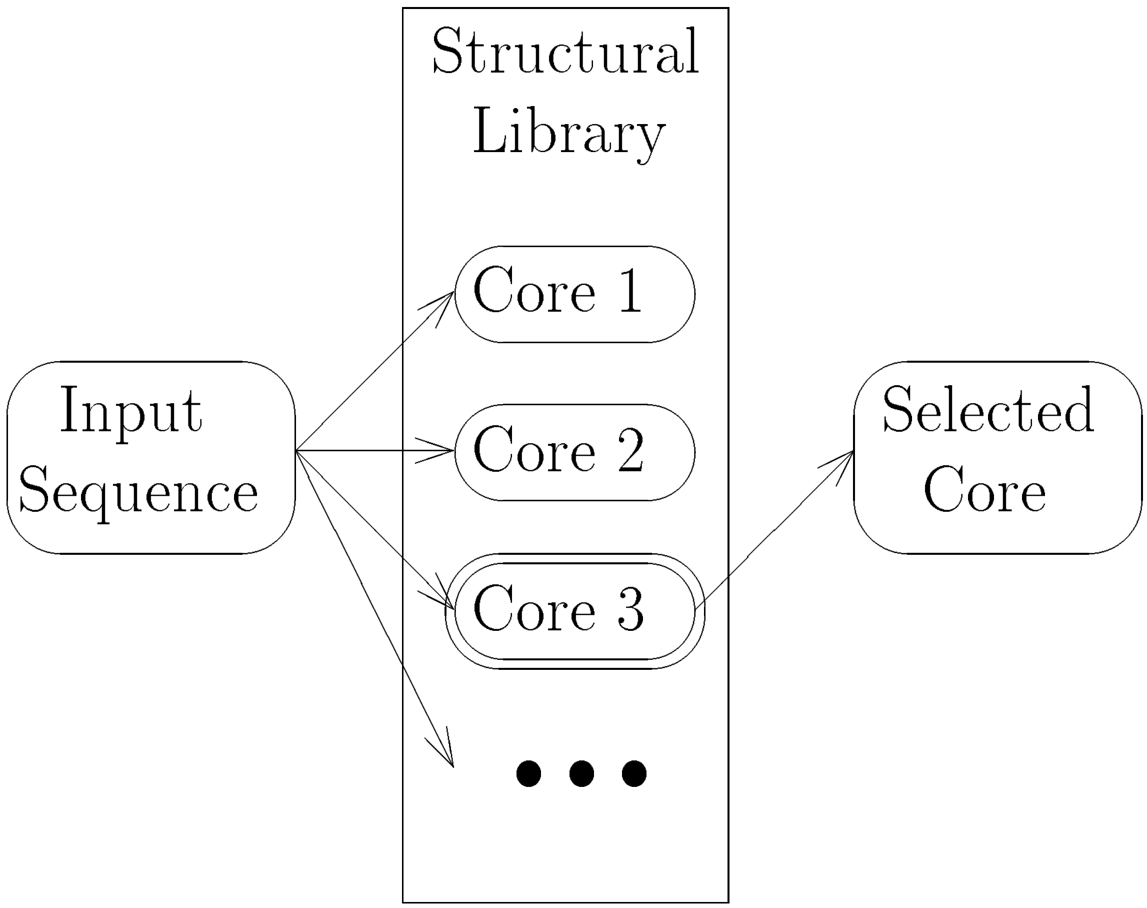
Structural
Library
Core 1
Core 2
Core 3
Input
Sequence
Selected
Core

Figure 1:

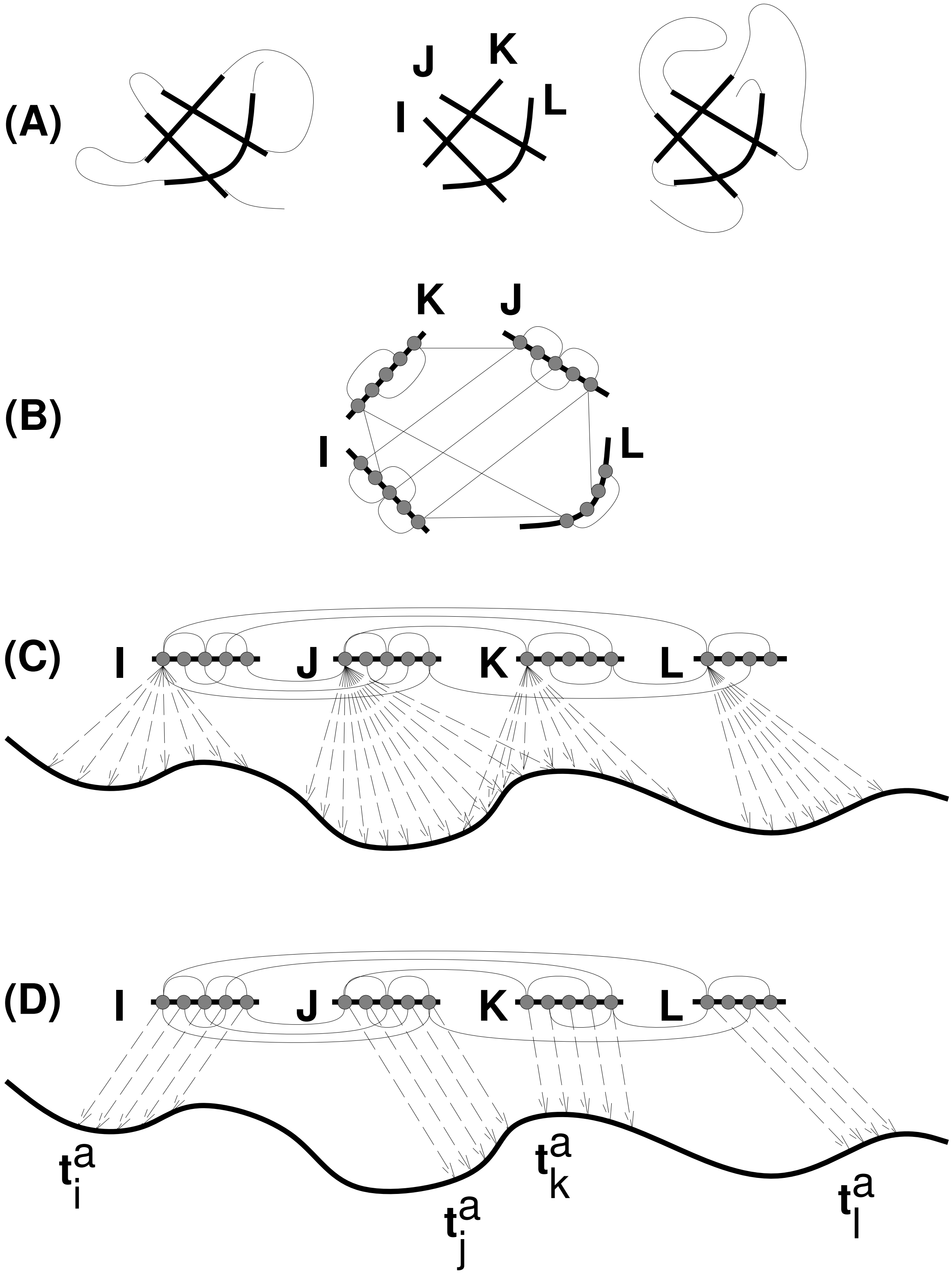

Figure 2:

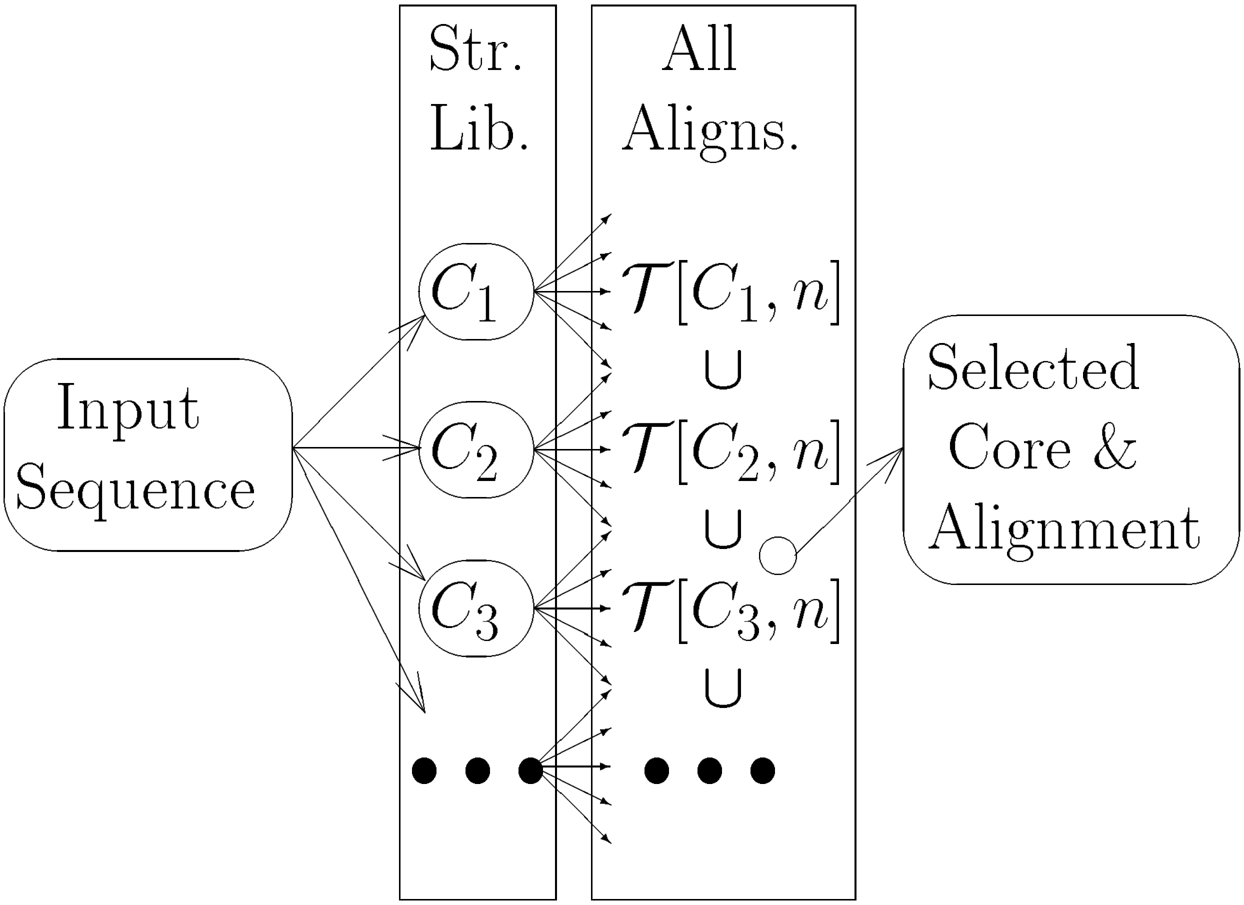
Str.
Lib.
All
Aligns.
Input
Sequence
C_1
C_2
C_3
$\mathcal{T}[C_1, n]$
$\cup$
$\mathcal{T}[C_2, n]$
$\cup$
$\mathcal{T}[C_3, n]$
$\cup$
Selected
Core &
Alignment

Figure 3:

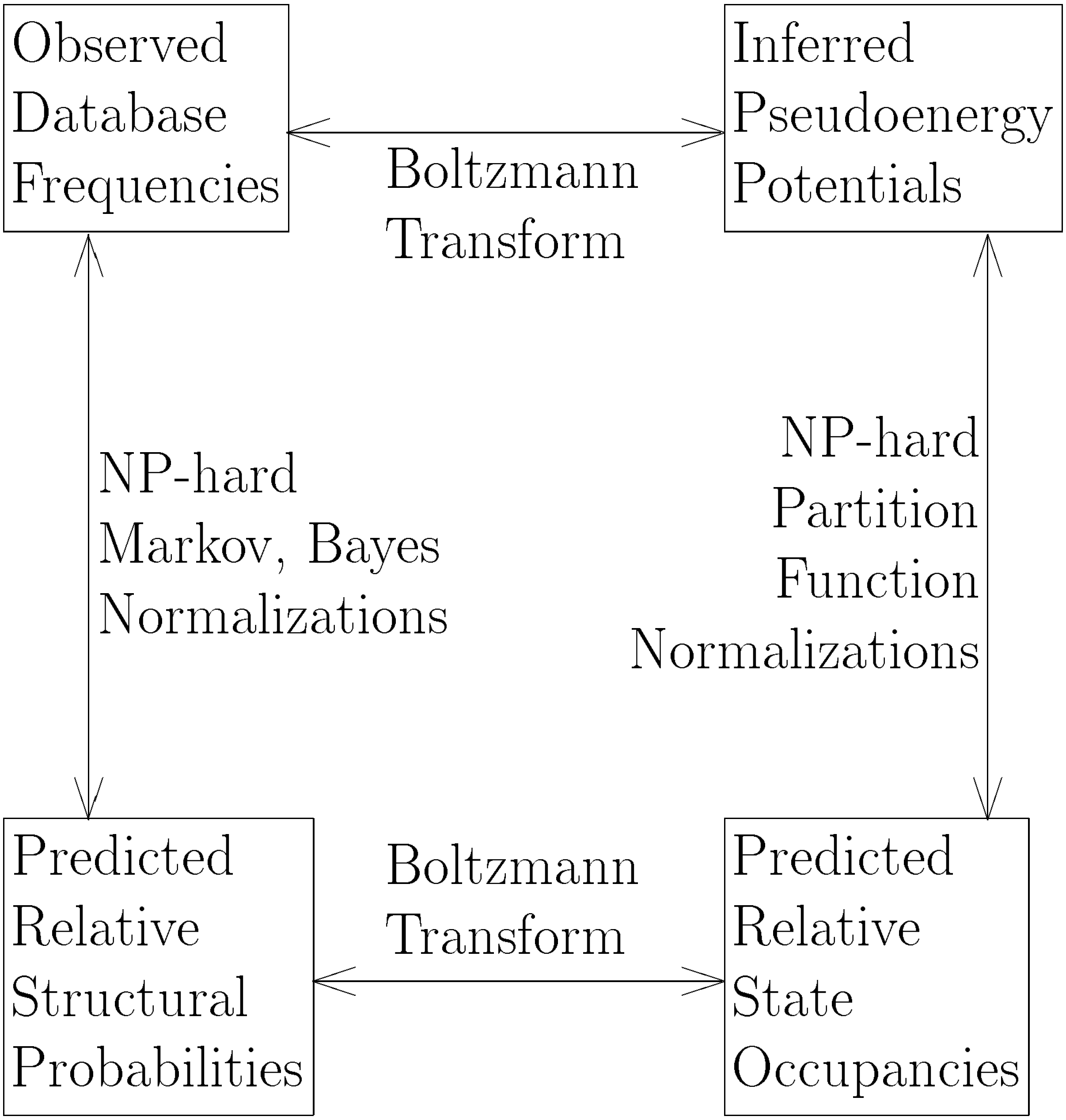
Observed Database Frequencies
Inferred Pseudoenergy Potentials
Boltzmann Transform
NP-hard Markov, Bayes Normalizations
NP-hard Partition Function Normalizations
Predicted Relative Structural Probabilities
Boltzmann Transform
Predicted Relative State Occupancies

Figure 4:

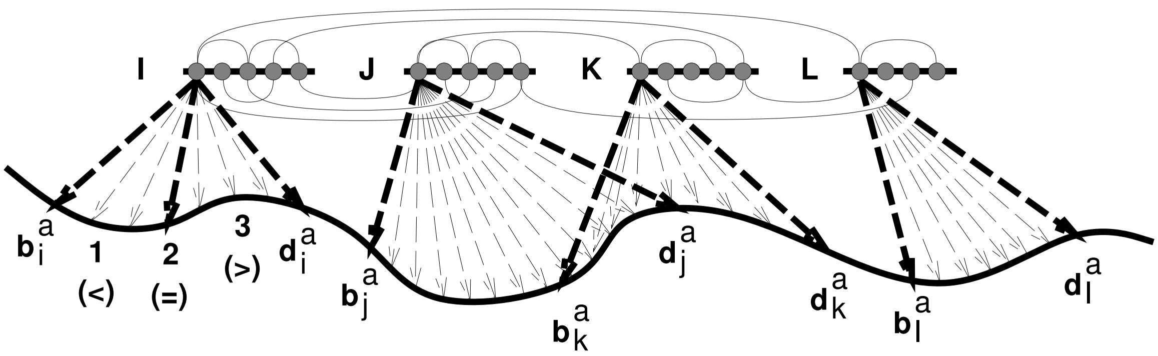

Figure 5:

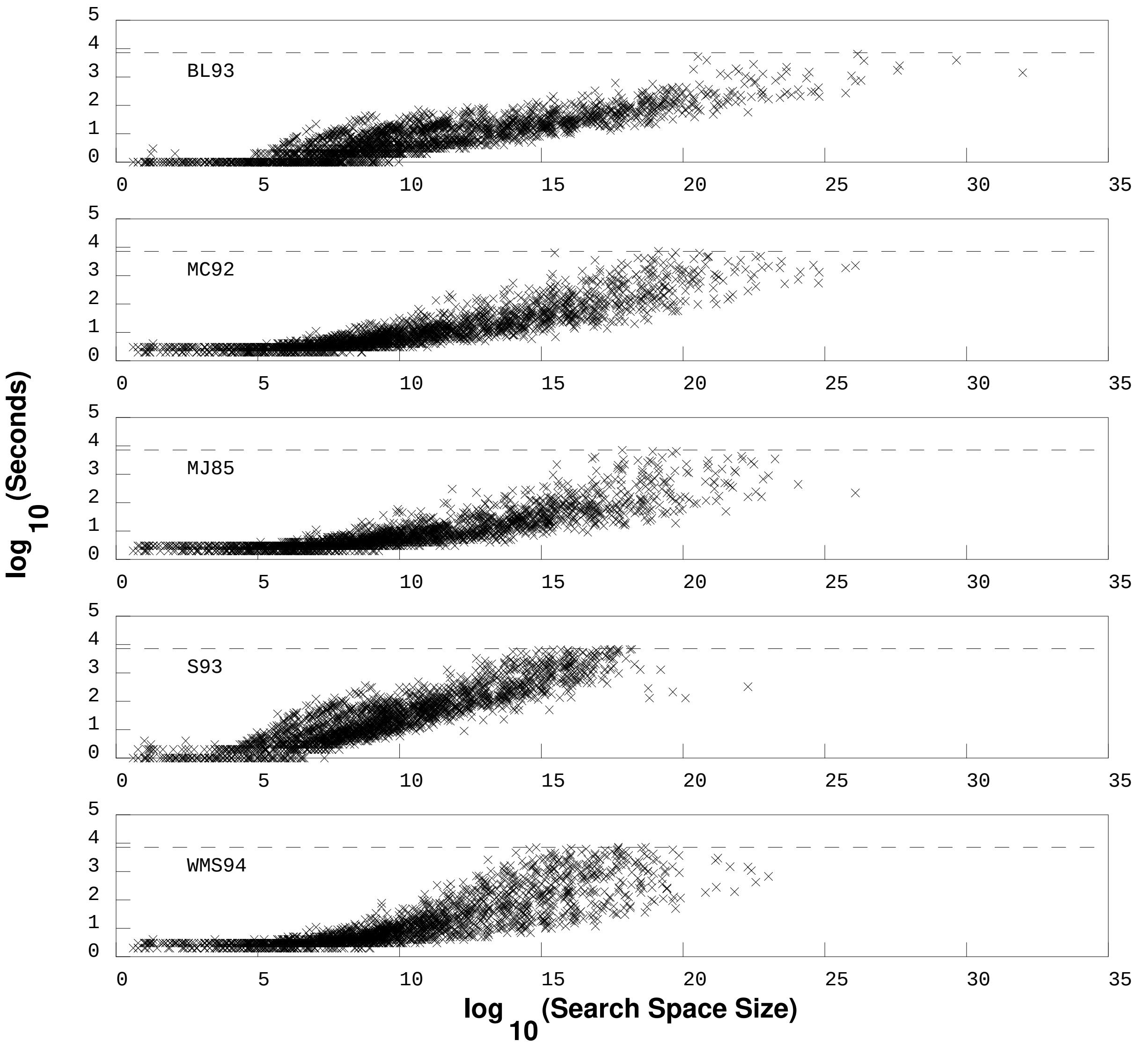

Figure 6: Slow Exponential Growth.

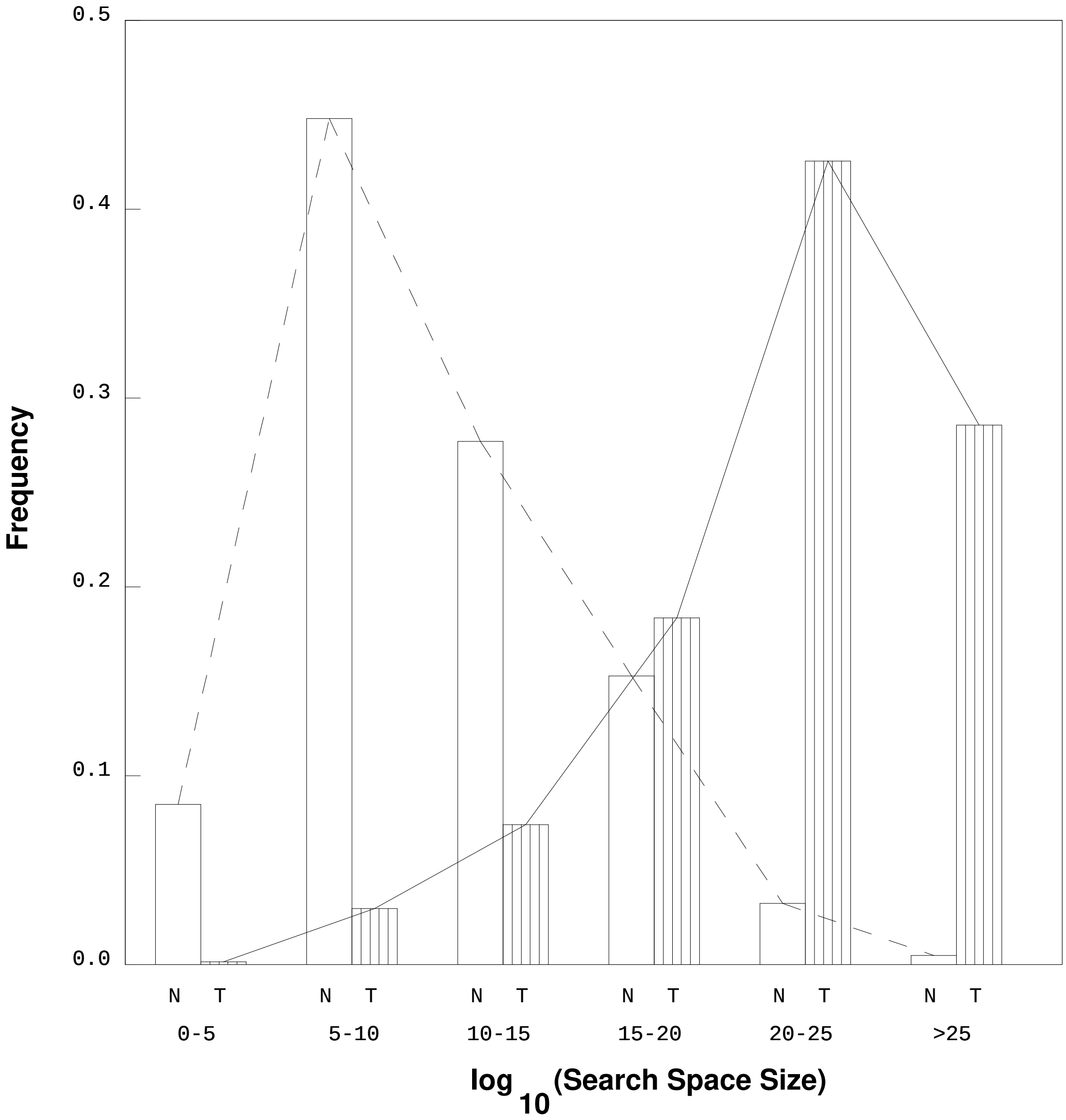

Figure 7: Histograms of number of trials ("N") and total time expended ("T").

Notation	Usage
$\mathbf{a}$	a sequence or string over A of length n
A	an alphabet of 20 characters (amino acid types)
A^n	the set of all strings over A of length n
$\mathbf{b}$ (or $\mathbf{d}$)	a vector of m integers; segment lower (or upper) bounds
c_i	$\lvert C_i \rvert$, the length of the i^{th} core segment C_i
C	a core structure; its i^{th} segment is C_i, whose j^{th} element is $C_{i,j}$
f	an objective function or score function
f^1	a sequence singleton-only version of f
f^A	a per-residue version of f^1
f_a (or f_{v}, or f_{e})	f restricted to amino acid residue types (or vertices, or edges)
f_l	f restricted to loops or variable regions
f_s	f restricted to core segments
$\overline{f}$	a mean or expected value of f
g	a per-segment encapsulation of f
h	$\exp(-f)$
h_λ	the loop length prior probability
H	$\sum_{\mathbf{w} \in A^k} h$
l_i (or l_i^{min}, or l_i^{max})	$\lvert \lambda_i \rvert$, the variable (or minimum, or maximum) length of the i^{th} loop λ_i
$\tilde{l}_i$	$l_i^{max} - l_i^{min}$, the variability of l_i
lb	a function returning a lower bound on scores achievable within a set
$\mathcal{L}$	a library of core structures
m	$\lvert C \rvert$, the number of core segments in C
n	$\lvert \mathbf{a} \rvert$, the length of the sequence $\mathbf{a}$
$\tilde{n}$	$n + 1 - \sum_i (c_i + l_i^{min})$, the relative sequence length
$P(A \vert B)$	the conditional probability of A given B
$P_1(i, t_i)$ (or $P_2(i, j, t_i, t_j)$)	the probability that a random threading places C_i at t_i (and C_j at t_j)
q, r	the inactive and active components of lb
s	a function returning a structural environment label
S, $S[\mathbf{b}, \mathbf{d}]$, $S\langle i, t_i \rangle$	the sizes of $\mathcal{T}[C, n]$, $\mathcal{T}[\mathbf{b}, \mathbf{d}]$, $\mathcal{T}\langle i, t_i \rangle$
$\mathbf{t}$ (or $\mathbf{t}^a$)	a vector of m integers; t_i (or t_i^a) is the i^{th} relative (or absolute) coordinate
$\mathcal{T}$	a set of alignments
$\mathcal{T}[C, n]$	the set of all alignments between core C and any sequence of length n
$\mathcal{T}[\mathbf{b}, \mathbf{d}]$	the set of alignments satisfying $b_i \leq t_i \leq d_i$
$\mathcal{T}\langle i, t_i \rangle$	the set of alignments that place C_i at t_i
V	the variance of a search space score distribution
$\mathbf{w}$ (or $\mathbf{x}$)	a summation variable over A^n (or over $\mathcal{T}[C, n]$)
Z, $Z_{\langle x,y,z \rangle}$	a partition function; a global sum specified by x, y, and z
$\alpha(i)$ (or $\beta(i, j)$)	an indicator of whether axis i (or either axis i or j) is active
$\mathcal{E}$	a set of adjacency graph edges, e or $\{u, v\}$; a subset of $\mathcal{V} \times \mathcal{V}$
λ	a set of loops; the i^{th} loop is λ_i, whose j^{th} element is $\lambda_{i,j}$
$\mu_{\langle x,y,z \rangle}$	a global mean specified by x, y, and z
σ	the standard deviation of a search space score distribution
$\mathcal{V}$	a set of adjacency graph vertices, v, corresponding bijectively to $\{C_{i,j}\}$
B, J, J^*, K, K^*, Q, Q_j, R	recurrence functions

Table 1: Notational usage of this paper.

PDB Code	Protein Length	Number of Core Segments	Search Space Size	Number of Search Iterations	Total (Search-only) Seconds	Equivalent Threadings per Iteration	Equivalent Threadings per Second
256b	106	5	6.19e+3	6	1 (1)	1.03e+3	6.19e+3
1end	137	3	4.79e+4	6	1 (1)	7.98e+3	4.79e+4
1rcb	129	4	5.89e+4	7	1 (1)	8.41e+3	5.89e+4
2mhr	118	4	9.14e+4	7	1 (1)	1.31e+4	9.14e+4
351c	82	4	1.12e+5	5	1 (1)	2.24e+4	1.12e+5
1bgc	174	4	1.63e+5	6	1 (1)	2.72e+4	1.63e+5
1ubq	76	5	1.70e+5	6	1 (1)	2.83e+4	1.70e+5
1mbd	153	8	1.77e+5	10	1 (1)	1.77e+4	1.77e+5
1lis	136	5	5.02e+5	7	1 (1)	7.17e+4	5.02e+5
1aep	161	5	5.76e+5	13	1 (1)	4.43e+4	5.76e+5
1hoe	74	6	7.36e+5	8	1 (1)	9.20e+4	7.36e+5
2hpr	87	6	1.34e+6	8	1 (1)	1.68e+5	1.34e+6
5cyt	103	5	1.37e+6	8	1 (1)	1.71e+5	1.37e+6
1bp2	123	5	1.53e+6	8	1 (1)	1.92e+5	1.53e+6
1aba	87	7	1.95e+6	13	1 (1)	1.50e+5	1.95e+6
1cew	108	6	2.32e+6	8	1 (1)	2.91e+5	2.32e+6
5cpv	108	5	2.60e+6	6	1 (1)	4.33e+5	2.60e+6
2mcm	112	10	1.31e+7	15	1 (1)	8.75e+5	1.31e+7
5fd1	106	5	2.25e+7	12	1 (1)	1.88e+6	2.25e+7
1plc	99	6	3.63e+7	10	1 (1)	3.63e+6	3.63e+7
1alc	123	6	1.70e+8	10	2 (1)	1.70e+7	8.51e+7
1yat	113	7	2.03e+8	8	1 (1)	2.54e+7	2.03e+8
7rsa	124	10	2.54e+8	12	1 (1)	2.12e+7	2.54e+8
3fxn	138	9	7.09e+8	12	2 (1)	5.91e+7	3.54e+8
9rnt	104	8	7.53e+8	21	2 (1)	3.58e+7	3.76e+8
2sns	149	8	2.19e+9	14	4 (1)	1.56e+8	5.47e+8
1ifc	132	12	2.31e+9	87	2 (1)	2.66e+7	1.16e+9
2lzm	164	12	3.16e+9	37	2 (1)	8.54e+7	1.58e+9
3chy	128	10	4.08e+9	45	1 (1)	9.06e+7	4.08e+9
1pkp	150	9	5.32e+9	20	3 (1)	2.66e+8	1.77e+9

Table 2: Timing details for self-threading (Bryant and Lawrence (1993), on DEC Alpha), part 1.

PDB Code	Protein Length	Number of Core Segments	Search Space Size	Number of Search Iterations	Total (Search-only) Seconds	Equivalent Threadings per Iteration	Equivalent Threadings per Second
1aak	152	8	2.34e+10	10	3 (1)	2.34e+9	7.82e+9
8dfr	189	10	1.45e+11	25	7 (1)	5.78e+9	2.06e+10
1cde	212	13	1.51e+11	38	5 (1)	3.99e+9	3.03e+10
2cpl	165	10	1.82e+11	17	5 (1)	1.07e+10	3.65e+10
3adk	194	13	1.89e+12	66	3 (1)	2.86e+10	6.30e+11
1rec	201	10	3.54e+12	30	4 (1)	1.18e+11	8.85e+11
2cyp	294	10	3.55e+12	181	20 (4)	1.96e+10	1.78e+11
1f3g	161	16	5.17e+12	45	6 (1)	1.15e+11	8.61e+11
4fgf	146	12	1.06e+13	48	4 (1)	2.22e+11	2.66e+12
1baa	243	9	1.53e+13	64	10 (2)	2.39e+11	1.53e+12
2act	220	11	1.12e+14	34	7 (1)	3.30e+12	1.60e+13
1dhr	241	14	4.56e+14	51	5 (1)	8.94e+12	9.12e+13
1mat	264	11	5.25e+14	100	15 (2)	5.25e+12	3.50e+13
1tie	172	12	1.19e+15	394	20 (9)	3.03e+12	5.96e+13
3est	240	13	1.92e+15	1946	47 (36)	9.85e+11	4.08e+13
2ca2	259	10	4.51e+15	100	20 (2)	4.51e+13	2.25e+14
1byh	214	14	1.07e+16	95	12 (4)	1.12e+14	8.90e+14
1apa	266	14	3.56e+17	141	18 (6)	2.52e+15	1.98e+16
4tgl	269	14	5.86e+18	361	22 (7)	1.62e+16	2.66e+17
5tmn	316	14	6.51e+18	164	28 (7)	3.97e+16	2.32e+17
1lec	242	15	7.01e+18	320	26 (12)	2.19e+16	2.70e+17
1nar	290	17	2.33e+19	3984	208 (183)	5.85e+15	1.12e+17
1s01	275	15	4.36e+19	541	32 (13)	8.05e+16	1.36e+18
5cpa	307	16	1.22e+20	1089	72 (50)	1.12e+17	1.69e+18
9api	384	17	1.95e+22	290	57 (25)	6.71e+19	3.41e+20
2had	310	19	2.57e+22	4027	201 (179)	6.39e+18	1.28e+20
2cpp	414	20	6.37e+24	3068	205 (164)	2.08e+21	3.11e+22
6taa	478	23	9.63e+31	4917	1409 (1267)	1.96e+28	6.83e+28

Table 2: Timing details for self-threading (Bryant and Lawrence (1993), on DEC Alpha), part 2.

Potentials	Native or Non-Native	Searches Converged		Total Hours	Avg. Secs. per Search	Regression Slope	Regression Interc.
BL93	Native	100%	(58/58)	0.7	43.3	0.12	-0.74
	Non-native	99.8%	(2467/2473)	42.3	61.5	0.13	-0.47
MC92	Native	98%	(57/58)	3.2	199.1	0.13	-0.50
	Non-native	98%	(2426/2473)	167.5	243.8	0.14	-0.44
MJ85	Native	98%	(57/58)	3.3	204.5	0.12	-0.42
	Non-native	99%	(2446/2473)	101.6	148.0	0.13	-0.35
S93	Native	90%	(52/58)	13.7	853.1	0.16	-0.43
	Non-native	89%	(2189/2473)	749.7	1091.4	0.24	-0.81
WMS94	Native	88%	(51/58)	18.4	1143.7	0.18	-0.74
	Non-native	93%	(2306/2473)	455.6	663.2	0.18	-0.62
Pooled	Pooled	96%	(12109/12655)	1556.0	442.6	0.15	-0.40

Table 3: Convergence rates, total hours, slopes, and intercepts.