

An Anytime Local-to-Global Optimization Algorithm for Protein Threading in $\mathcal{O}(m^2\tilde{n}^2)$ Space

RICHARD H. LATHROP

ABSTRACT

This paper describes a novel anytime branch-and-bound or best-first threading search algorithm for gapped block protein sequence–structure alignment with general sequence residue pair interactions. The new algorithm (1) returns a good approximate answer quickly, (2) iteratively improves that answer to the global optimum if allowed more time, (3) eventually produces a proof that the final answer found is indeed the global optimum, and (4) always terminates correctly within a bounded number of steps if allowed sufficient space and time. It runs in polynomial space, which is asymptotically dominated by the $\mathcal{O}(m^2\tilde{n}^2)$ space required by the lower bound computation. Using previously published data sets and the Bryant–Lawrence (1993) objective function, the algorithm found the true (proven) global optimum in less than 5 min in all search spaces size 10^{25} or smaller (sequences to 478 residues), and a putative (not guaranteed) optimum in less than 5 hr in all search spaces size 10^{60} or smaller (sequences to 793 residues, cores to 42 secondary structure segments). The threading in the largest case studied was eventually proven to be globally optimal; the corresponding search speed in that case was the equivalent of 1.5×10^{56} threadings/sec, a speed-up exceeding 10^{25} over previously published batch branch-and-bound speeds, and exceeding 10^{50} over previously published exhaustive search speeds, using the same objective function and threading paradigm. Implementation-independent measures of search efficiency are defined for equivalent branching factor, depth, and probability of success per draw; empirical data on these measures are given. The general approach should apply to other alignment methodologies and search methods that use a divide-and-conquer strategy.

Key words: protein threading, inverse folding, fold recognition, sequence, structure, alignment, pair potentials, contact potentials, knowledge-based potentials, search.

1. INTRODUCTION

PROTEIN STRUCTURE PREDICTION from sequence is one of the great unsolved challenges of molecular biology. Protein sequence–structure alignment, also called protein threading, attempts to model a novel sequence using a known structure by aligning the sequence to the structure under an objective function. After alignment, the sequence is given a similar three-dimensional (3D) fold by assigning each sequence residue the 3D coordinates of the aligned structure residue. For reviews see Bowie and Eisenberg (1993), Bryant and Altschul (1995), Fetrow and Bryant (1993), Jernigan and Bahar (1996), Jones and Thornton (1993), Jones

and Thornton (1996), Lemer *et al.* (1995), Sippl (1995), and Wodak and Rooman (1993), while for cautionary notes see Crippen (1996), Lathrop and Smith (1996), Moulton *et al.* (1995), Ouzounis *et al.* (1993), Russell and Barton (1994), Smith *et al.* (1997), and Thomas and Dill (1996).

The protein sequence–structure alignment problem consists of a sequence, a structure [considered as drawn from a library of structures (Lathrop *et al.*, 1998)], a set of legal sequence–structure alignments, and an objective function that maps each legal alignment to a real number. The objective function value is the score, pseudoenergy, or potential of any given alignment; here, by analogy to energy, lower numbers are more favorable. The sequence, structure, and objective function together determine the entire alignment landscape including the locations and values of all global and local minima. The accuracy of the objective function is measured by the agreement (or lack of it) between crystallographic alignments and the objective function alignment minima (Fischer *et al.*, 1996).

1.1. Search algorithms

For purposes of this paper, assume that the protein sequence, structure, objective function, and set of legal alignments all are fixed in advance. This fixes the alignment landscape and produces a computational optimization task: to find the alignments within the search space that minimize the objective function. The search space is the set of legal alignments, the search space size is their cardinality, and the search goal is to find alignments that minimize the objective function. This optimization task is the primitive computational step in many approaches to protein threading by sequence–structure alignment.

The objective function determines the threading landscape and whether it matches crystallographic alignments, while the search algorithm determines whether the alignments that are found actually match the landscape minima indicated by the objective function. Consequently, search algorithms are evaluated on how well and how quickly they can find the objective function alignment minima. Search accuracy is measured by agreement (or lack of it) between the objective function minima and alignments returned by the search algorithm. Efficiency is measured by the amount of computational resources expended. In comparisons, it is important to use a previously characterized objective function and a large diverse set of sequences and structures because search efficiencies vary considerably across these. The Bryant–Lawrence (1993) objective function (Bryant and Lawrence, 1993) was the fastest of five objective functions studied (Lathrop and Smith, 1996). Speeds have been reported of 1.4×10^2 threadings/sec for exhaustive search (Bryant and Lawrence, 1993) and 6.8×10^{28} for branch-and-bound search (Lathrop and Smith, 1996), both on faster machines than here.

1.1.1. Categorization. Exhaustive search could accomplish the optimization task using a number of objective function evaluations equal to the search space size, but the search space becomes combinatorically large and so other algorithms have been sought. Protein threading search algorithm classes depend on computational behavior and assumptions about protein structure:

- **Pair Interactions: Modeled vs. Not.** If interactions between pairs of sequence residues are not modeled then the problem is low-order polynomial using dynamic programming (Sankof and Kruskal, 1983). In contrast, if both alignment gaps and pair interactions between sequence residues are modeled, then restricted subproblems may be polynomial of varying degree (Xu and Uberbacher, 1996; Xu *et al.*, 1998) while the general pair interaction problem treated here is NP-hard (Akutsu and Miyano, 1997; Lathrop, 1994).
- **Alignment: Gaps Anywhere vs. Gapped Block.** Alignment gaps may be permitted anywhere in the structure, or in contrast may be confined to loop regions between discrete blocks of core secondary structure segments. Permitting gaps anywhere may introduce unphysical mainchain breaks in the interior of secondary structure segments, while confining gaps to the loops may produce erroneous secondary structure segment lengths. Gaps anywhere results in much larger search space sizes than gapped block.
- **Global Optimality: Exact (Proven) vs. Approximate (Not Guaranteed).** An exact algorithm produces an implicit proof that the answer found is indeed the global optimum. In contrast, an approximate algorithm produces an answer quickly, but without proof of global optimality; its optimality, if any, is unknown.
- **Answers: Anytime vs. Batch.** An anytime algorithm produces a good answer quickly, then iteratively improves that answer if allowed more time. After a short initialization period it may be stopped at any time and a usable answer obtained. Approximate algorithms that rely on iterative sampling are anytime algorithms because they may be stopped at any time and return their current best candidate. In contrast, a batch algorithm produces no answer until the end of the run, and if stopped early produces no answer at all.

Examples of these categories may be drawn from the literature; the discussion here assumes that pair interactions are modeled. Within the gaps anywhere paradigm, examples of approximate algorithms include the batch double dynamic programming algorithm (Taylor and Orengo, 1989) and the anytime frozen approximation algorithm (Godzik *et al.*, 1992). Within the gapped block paradigm, examples of approximate algorithms include the anytime Gibbs Sampler (Lawrence *et al.*, 1993; Madej *et al.*, 1995) and the stochastic algorithm of Crawford (1999). Examples of exact algorithms include exhaustive search (Bryant and Lawrence, 1993), batch branch-and-bound (Lathrop and Smith, 1996), and batch divide-and-conquer (Xu and Uberbacher, 1996; Xu *et al.*, 1998).

1.2. This paper

The algorithm described in this paper is a gapped block alignment algorithm that models fully general sequence residue pair interactions in the objective function. It is an anytime algorithm that is initially approximate but becomes exact after a bounded number of steps. It runs in polynomial space, which is asymptotically dominated by the space required by the lower bound computation, $\mathcal{O}(m^2\tilde{n}^2)$. The algorithmic strategy employed is to adapt a batch branch-and-bound threading algorithm (Lathrop and Smith, 1996) to employ a best-first anytime control structure. The result is a branch-and-bound or best-first threading algorithm that returns (1) a good answer quickly, (2) the true global optimum shortly thereafter in all cases we have been able to verify, and (3) eventually, a proof of optimality if allowed sufficient time to run to completion.

1.2.1. Batch branch and bound background. The method presented here is a modified version of a batch branch-and-bound threading algorithm (Lathrop and Smith, 1996). An alignment may be represented as a vector \mathbf{t} , where t_i gives the sequence residue aligned to the i th structure coordinate. The vector space dimension is m , the sequence length is n , and the number of distinct sequence residues that may align to a given structure coordinate is \tilde{n} . The hyperrectangle $[\mathbf{b}, \mathbf{d}]$, whose corners are the vectors \mathbf{b} and \mathbf{d} , contains all alignments \mathbf{t} such that $b_i \leq t_i \leq d_i$. Each hyperrectangle corresponds to a set of partially instantiated alignments, and its vector subspace dimension corresponds to the number of unaligned core segments. A lower bound of complexity $\mathcal{O}(m^2\tilde{n}^2)$ maps a hyperrectangle to a lower bound on the possible values of the objective function on any point in the hyperrectangle. A splitting function partitions a hyperrectangle into a small set of mutually disjoint and exhaustive smaller hyperrectangles, at least one of which is of lower vector subspace dimension than the parent. A priority queue (or heap) holds a sorted list of all currently instantiated hyperrectangles, sorted by the lower bound.

Initially the queue holds a single hyperrectangle, $[\mathbf{1}, \tilde{\mathbf{n}}]$, that covers the entire search space. At each step, the hyperrectangle having the currently lowest lower bound is removed from the queue. If it contains only one alignment it is returned as a global optimum, because no other hyperrectangle on the queue can possibly achieve a lower score. Otherwise it is partitioned using the splitting function, and the resulting smaller hyperrectangles are merged into the queue ordered by the lower bound.

2. METHODS

First a new method of searching for threading landscape minima is described. Then implementation and hardware details are given. Next the data sets, objective function, and run conditions are stated. Finally, several implementation-independent measures of search efficiency are defined.

2.1. A local-to-global anytime search method

In this paper, the original algorithm (Lathrop and Smith, 1996) is modified to use $m + 1$ priority queues. Queue Q_k holds hyperrectangles of vector subspace dimension $m - k$. The hyperrectangles in each queue are sorted by their lower bound. Figure 1 gives a schematic illustration. The queues are arranged in a cascade (left-to-right in Fig. 1) according to the number of aligned core segments, i.e., in decreasing order of hyperrectangle dimension. The idea is that if the lower bound is not too loose, then hyperrectangles that enclose favorable threadings will tend to sort to the front of the queues in Fig. 1 and from there migrate to the more fully instantiated higher-numbered queues. The Appendix gives pseudocode for the control architecture described in this section.

When a hyperrectangle is split, at least one resulting hyperrectangle is guaranteed to decrease in dimension and so will move to a higher-numbered queue (arrows in Fig. 1). Consequently, it is always possible to “sweep” across the queues by beginning at the lowest-numbered occupied queue and at each step: (1) popping the best

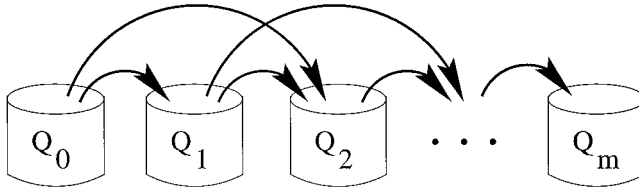


FIG. 1. Schematic illustration of anytime queue cascade. Queue Q_0 holds completely uninstantiated alignments, queue Q_m holds fully instantiated alignments, and intermediate queues Q_k hold partial alignments corresponding to hyperrectangles of dimension $m - k$. Here, queue Q_k holds partial alignments with exactly k aligned (fixed) core segments. In a gap-anywhere paradigm, Q_k might hold partial alignments with k aligned residues. When a partial alignment is removed from one queue and partitioned, the resulting partial alignments are reinserted into new queues corresponding to their vector subspace dimension. If the dimension has not changed they are reinserted into the same queue (not shown); otherwise they are reinserted into the appropriate queue holding more completely instantiated alignments (arrows).

partial alignment from the top of the current queue, (2) splitting it and reinserting the children into queues as appropriate, and (3) advancing to the next highest-numbered occupied queue. One such sweep can be done in $\mathcal{O}(m)$ lower bound evaluations, and is guaranteed to produce a fully instantiated alignment from Q_m at the end.

In addition to sweeping across the queues, it is possible to operate opportunistically on the queue that currently appears to contain the most promising possibilities. The basic idea is simple. Ignoring outliers beyond some z -score threshold above the current mean, the algorithm estimates internally the recent (decaying average) mean and standard deviation of the increase in lower bound that previous splits have achieved when moving from queue Q_i to queue Q_{i+1} . As described below, these yield a very crude estimate of the probability that the top hyperrectangle in each queue contains an alignment that scores better than the current anytime best. The hyperrectangle with the highest crude estimate is chosen and split.

Although best-first search usually requires both exponential time and exponential space, a polynomial space bound can be achieved for the task considered here; the time bound, however, remains exponential. The user specifies an arbitrary space constant, C (currently 100,000 in our implementation). A second set of $m + 1$ priority queues is used, called the annihilation queues AQ , which mirror the primary queues Q described above. Annihilation queue AQ_k also holds hyperrectangles of vector subspace dimension $m - k$, also sorted by their lower bound. Whenever the total number of hyperrectangles on the primary queues exceeds C , one hyperrectangle is removed from the primary queues and placed in its mirroring annihilation queue. Since each split consumes one hyperrectangle and produces at most three, the number of hyperrectangle in the primary queues can never exceed $C + 2$. Whenever any of the annihilation queues contain any hyperrectangles, processing of the primary queues is suspended and only hyperrectangles in the annihilation queues are processed until the annihilation queues again are empty. The result is that the total number of hyperrectangles in the primary queues is decreased by one. Annihilation queues are processed by choosing the highest-numbered occupied queue, removing and splitting its top hyperrectangle, and inserting its children into the appropriate annihilation queues. In this process, at most two children are returned to the same annihilation queue from which the parent was removed, while at most three children are promoted to higher-numbered annihilation queues. One hyperrectangle ultimately can generate at most $\tilde{n}/3 + 1$ descendants that are returned to its same annihilation queue, and so there can be no more than $\tilde{n} + 3$ hyperrectangles in any annihilation queue. Since there are $m + 1$ annihilation queues, the total number of hyperrectangles on all queues is $\mathcal{O}(C + m\tilde{n})$. Each hyperrectangle requires $2m$ integers, so the total space complexity of all queues is $\mathcal{O}(Cm + m^2\tilde{n})$. This is dominated by the lower bound space complexity, which is $\mathcal{O}(m^2\tilde{n}^2)$. Consequently, the entire algorithm runs in $\mathcal{O}(m^2\tilde{n}^2)$ space.

2.1.1. Overall control architecture. Suppose the user wishes to return the K best threadings in the search space. Normally $K = 1$, indicating to return only the global optimum (called the “anytime best” below), but sometimes near-optimal alignments are of interest too. The overall control architecture begins with at least K sweeps to generate the initial K candidates, thereafter runs in opportunistic mode in an attempt to improve them, and temporarily shifts to annihilation mode whenever the total number of hyperrectangles exceeds the user-specified space parameter C . First the queues are initialized. Q_0 holds a single hyperrectangle containing the entire search space, $[\mathbf{1}, \tilde{\mathbf{n}}]$, and all other queues are empty. Initially $\max(K, 5)$ sweeps are done in order to produce K good threadings quickly and to initialize the mean and standard deviation estimates. Thereafter the algorithm opportunistically operates on the most productive regions of the search space. Whenever a fully instantiated threading is produced that scores better than the worst scoring of the current K best candidates, it replaces that candidate.

The algorithm remembers its best K candidates so far, and presents this as its best guess if it is stopped before it terminates normally. Otherwise, if allowed sufficient time, eventually it reaches a state in which the top hyperrectangle of every queue has a lower bound that is equal to or greater than the actual score of the worst scoring of the current K best candidates. This constitutes an implicit proof that the current anytime best is in fact a global optimum, because no other hyperrectangle anywhere in the search space can possibly contain an alignment of lower score. When this occurs the algorithm announces that it has proven that the current K candidates are the globally optimal K best threadings, and then terminates normally.

2.1.2. An improved splitting function. The original batch splitting function (Lathrop and Smith, 1996) partitioned hyperrectangles according to a complicated function of partial probabilities over partial alignment scores. A simpler and more effective splitting function, used for all results presented in this paper, is as follows:

1. If one segment interacts with more different segments than any other, split on that segment.
2. Break ties by splitting on the segment that has the most total number of residue interactions.
3. Break the remaining ties by splitting on the segment furthest from any fixed segment.
4. Break the remaining ties by splitting on the segment closest to the middle of the structure.

A second splitting function, also more effective than the original (Lathrop and Smith, 1996), exploits the lower bound computation to guide the choice of split point. This chooses the segment whose split maximizes the lowest lower bound of the children, i.e., the maximum over all segments of the minimum over all children resulting from splitting on that segment of the lower bound of the child. If done at each split, this requires $3m$ lower bound computations to choose one split point, and is impractical. However, an acceptable static compromise results from doing the full computation only once, on the very first initialization sweep, and recording in a table the order in which segments were chosen to be split. Thereafter, splits are chosen by looking up the next split segment in the recorded table.

2.2. Implementation and hardware

The algorithm is implemented in Common Lisp. The results below were obtained on a 110-MHz SPARCstation 5 desktop workstation. For very large problem sizes the choice of algorithm completely dominates the choice of programming language or machine hardware as an influence on search speed.

2.3. Data sets, objective function, run conditions

The algorithm was run using the previously published objective function and threading paradigm of Bryant and Lawrence (1993). Data sets were taken from Lathrop and Smith (1996), Rost *et al.*, (1997), and a SCOP (Murzin *et al.*, 1995) core domain database. The data set of Lathrop and Smith (1996) has 60 sequences (length to 478 residues) and 60 cores (of 3 to 23 core segments) yielding search space sizes up to 9.4×10^{31} . The data set of Rost *et al.* (1997) has 16 sequences (length to 224 residues) and 16 cores (of 4 to 19 core segments) yielding search space sizes up to 2.1×10^{22} . In the SCOP (Murzin *et al.*, 1995) core domain database the five largest cores (39 to 42 core segments) were run against all 473 sequences (length to 793 residues) yielding search space sizes up to 2.1×10^{59} .

The algorithm was run to a proven global optimum on every trial in the data sets of Lathrop and Smith (1996) and Rost *et al.* (1997). On the sequences and cores from SCOP (Murzin *et al.*, 1995) the algorithm was stopped after it had generated 100 candidate alignments beyond the current anytime best without either improving its score or proving it optimal. Consequently, every anytime best in the data sets of Lathrop and Smith (1996) and Rost *et al.* (1997) is a true global optimum (proven), while many anytime bests in the SCOP largest cores data set are only putative (not guaranteed).

Other trials were run to compare the algorithm's behavior across different objective functions, as well as to compare the anytime branch-and-bound performance to the previous batch branch-and-bound (Lathrop and Smith, 1996). The algorithm was run on the data set of Lathrop and Smith (1996) using two other previously published objective functions (White *et al.*, 1994; Sippl, 1993). These objective functions, as well as that of Bryant and Lawrence (1993), have been characterized previously on this data set (Lathrop and Smith, 1996).

2.4. Implementation-independent metrics of search efficiency

The most important measure of search algorithm efficiency is the number of elapsed seconds, since this is how long we must wait for an answer. However, it is convenient to have other measures for algorithm comparison that are implementation independent.

Let S be the size of the search space and L be the number of primitive objective function or lower bound evaluations. Let the subscript “Exh” mean exhaustive search and “Alg” mean the search algorithm being compared. Below, “Alg” will be further specialized so that “Any” means the point at which the final anytime best was encountered and “Lim” means the point at which search was abandoned.

2.4.1. Branching factor. Suppose a uniform search tree of branching factor b and depth d , then there are b^d leaf nodes. For exhaustive search, $b^d = S$ and $d = m$, so the branching factor for exhaustive search is $b_{\text{exh}} = S^{1/m} = \sqrt[m]{S}$. The equivalent branching factor for the algorithm (Pearl, 1984) is $b_{\text{alg}} = L^{1/m} = \sqrt[m]{L_{\text{alg}}}$.

2.4.2. Search depth. Recent experimental results (Korf and Reid, 1998) suggest that perhaps the branching factor remains unchanged at b_{exh} , and that heuristic search instead reduces the effective search depth. In this case the search depth for exhaustive search is $d_{\text{exh}} = \log S / \log b_{\text{exh}} = m$, and $d_{\text{alg}} = \log L_{\text{alg}} / \log b_{\text{exh}}$.

2.4.3. Probability of success. Let P be the probability per draw without replacement of guessing the optimal in one blind guess. Then $P_{\text{exh}} = 1/S$ and $P_{\text{alg}} = 1/L_{\text{alg}}$.

3. RESULTS

The results below are adapted from a preliminary presentation in Lathrop (1999).

Figure 2A shows a histogram of the rank of the final anytime best in the list of candidates. The rank is 1 if it was the first candidate produced, 2 if the second, and so forth. Figure 2B shows a histogram of the maximum

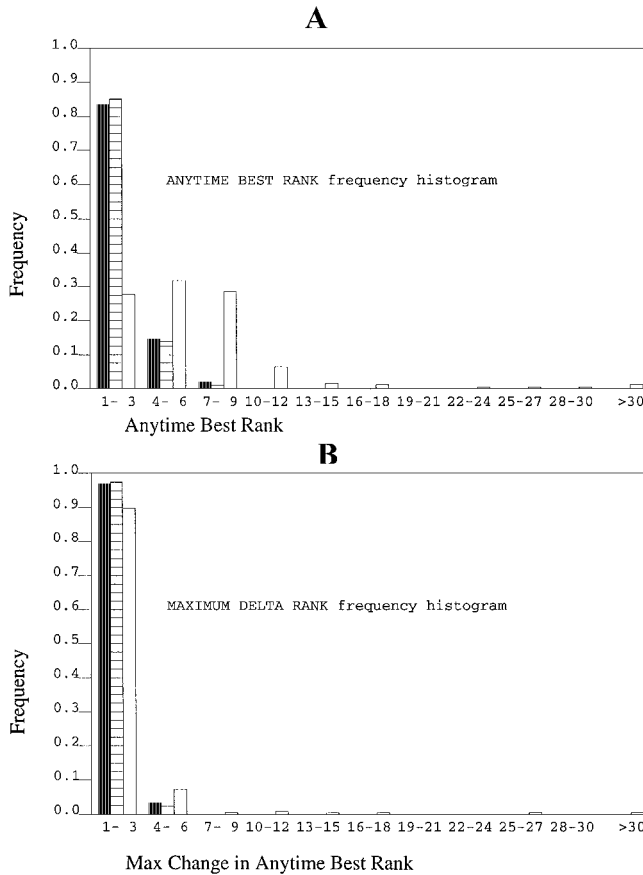


FIG. 2. Final anytime best rank and maximum Δ rank. **(A)** Histogram of the rank in the enumerated candidate alignments at which the final anytime best was found. **(B)** Histogram of the maximum change in the rank of the current anytime best at any time during the search. In both histograms, black is the data set of Lathrop and Smith (1996), horizontal stripes is Rost *et al.*, (1997), and white is the five largest cores from the SCOP domain database (Murzin *et al.*, 1995). For Lathrop and Smith (1996) and Rost *et al.* (1997) the final anytime best was proven to be the global optimum; for SCOP (Murzin *et al.*, 1995) search was abandoned after 100 candidates were enumerated without improvement.

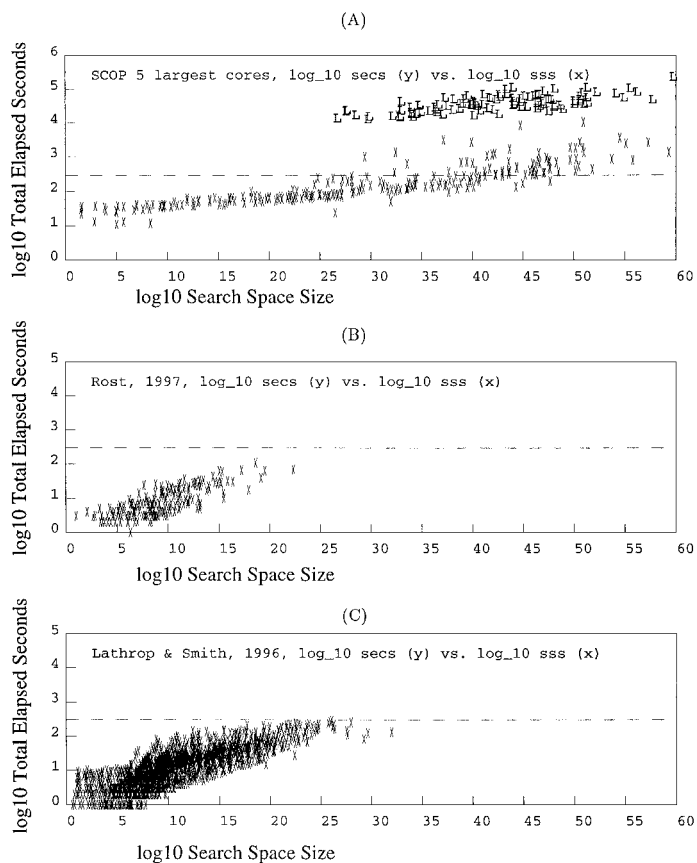


FIG. 3. \log_{10} total seconds to final anytime best. (A) The five largest cores from the SCOP data set (Murzin *et al.*, 1995). (B) The data set of Rost *et al.* (1997). (C) The data set of Lathrop and Smith (1996). In all graphs the x -axis is the \log_{10} of the search space size. The dashed line corresponds to 5 min. The runs correspond to Fig. 2. For each search, an “x” marks the \log_{10} of the total number of seconds to the final anytime best. In (A), an “L” marks the \log_{10} of the number of seconds at which search was abandoned if the final anytime best was not proven globally optimal (after 100 candidates without change). Absence of an “L” implies that proof of global optimality was obtained. The point corresponding to (32.6, 4.2) is a performance outlier in which the “x” is obscured by “L”s. In (B) and (C) search was always continued until the final anytime best was proven to be globally optimal.

Δ rank observed. The Δ rank is the rank of the current anytime best minus the rank of the previous anytime best that it replaced. That is, it is the number of candidates enumerated from one change of the anytime best to the next.

Figure 3 shows the total elapsed clock seconds until the final anytime best was produced vs. the search space size, on \log_{10} – \log_{10} axes (dashed line = 5 min). In Fig. 3A an “L” marks the time at which search was abandoned if the global optimum was not proven. In Fig. 3B and C the search was always continued until the global optimum was proven. Here it produced the true global optimum as its anytime best within 5 min of total elapsed time (dashed lines), although of course to produce the proof of optimality required additional time.

Figure 4 shows how measures derived from Figs. 2 and 3 depend on search space size. Figure 4A shows the rank of the final anytime best, and Fig. 4B shows the maximum Δ rank observed. Figure 4C shows the total elapsed seconds/ $m^2\bar{n}^2$, i.e., the total time divided by the formal complexity of the lower bound calculation, which is $\mathcal{O}(m^2\bar{n}^2)$.

Figure 5 shows the implementation-independent metrics of search efficiency described above, with exhaustive search compared to branch-and-bound. Figure 5A shows the branching factor b , Fig. 5B shows the search depth d , and Fig. 5C shows the probability of success per draw P .

Tables 1 and 2 provide detailed analyses of the five largest SCOP cores against the five longest SCOP sequences. Note that in approximately half (13/25) of the cases shown, the final anytime best was obtained in 15 min or less total elapsed time.

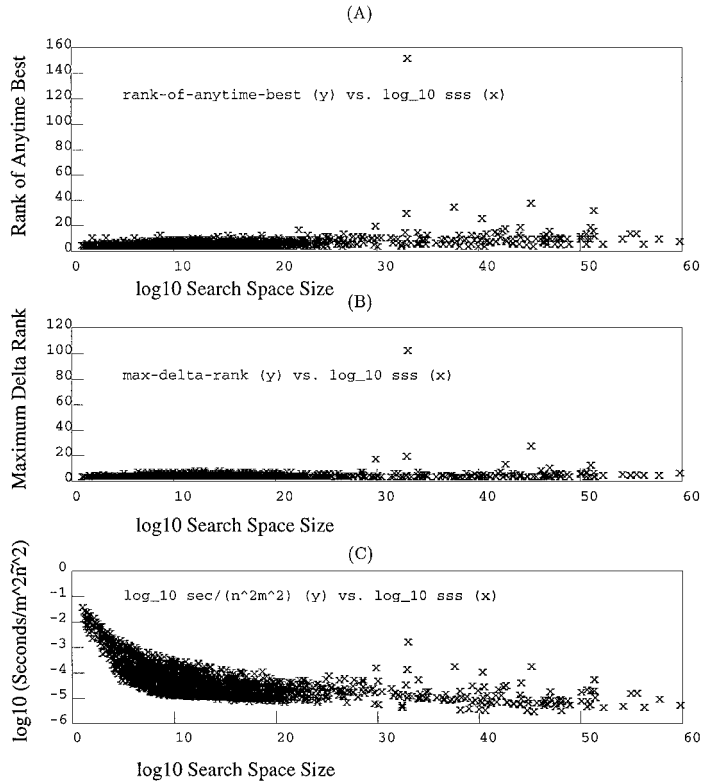


FIG. 4. Search space size dependencies. **(A)** Rank of final anytime best; the points correspond to Fig. 2A with all data sets pooled. **(B)** Maximum Δ rank; the points correspond to Fig. 2B with all data sets pooled. **(C)** \log_{10} [total seconds to final anytime best / ($\tilde{n}^2 m^2$)]; the points correspond to Fig. 3 with all data sets pooled and divided by the formal lower bound complexity, $\mathcal{O}(m^2 \tilde{n}^2)$. In all graphs, the x -axis is the \log_{10} of the search space size. The point at $x = 32.6$ corresponds to the performance outlier (32.6, 4.2) in Fig. 3A. By coincidence, its maximum Δ rank was 100, numerically equal to the Δ rank cut-off for abandoning the search.

Table 3 compares the three data sets across the same objective function, and three different previously characterized objective functions across the same data set. It provides mean rank and mean maximum change in rank of the anytime best, as well as slopes and intercepts of the best-fitting regression line to the log–log plot of elapsed time vs. search space size. Relative objective function difficulty, as measured by the slope of the regression line, is the same for the anytime best time, the anytime proof time, and the batch proof time. However, the slope for the anytime best is consistently less than the slope for the proofs.

4. DISCUSSION

The anytime branch-and-bound or best-first approach above has several useful characteristics. It appears to return the global optimum quickly in many but not all realistic cases. It can return good approximate alignments quickly when in an early exploratory mode, then later lock those in with more effort when a predictive effort is in its final stages. It has a user-adjustable space bound that permits space to be traded for time in a controllable way. Because the search space is explicitly represented and sampled without replacement, it never returns the same candidate twice.

It is plausible that many of the putative results on the SCOP trials marked “L” in Fig. 3A, for which the anytime best was not proven to be the global optimum, indeed eventually would be proven to be the global optimum if the program were allowed to run sufficiently long. They share many gross characteristics with the proven global optima in the smaller trials: their ranks, Δ ranks, search times, branching factors, search depths, and probabilities per draw are comparable. Indeed, the search was continued to the proof of global optimality on both the largest trial of Table 1 (No. 25, 1bgw vs. 1tsp) and the performance outlier of Fig. 3A (32.6, 4.2). In both cases the anytime best shown eventually was proven to be globally optimal, though the proof in the case of 1bgw vs. 1tsp required over 15 days of computer time (data not shown).

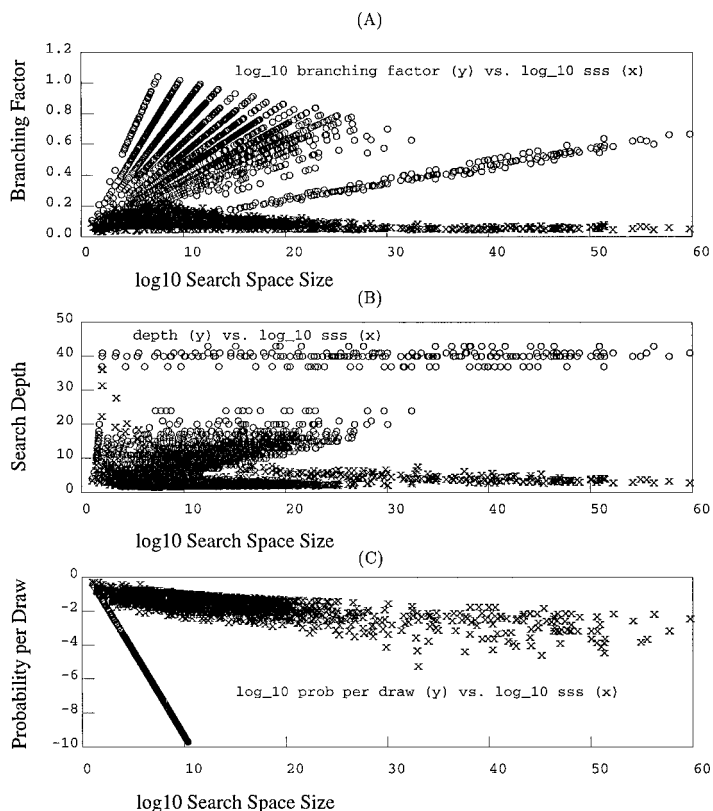


FIG. 5. Implementation-independent metrics of search efficiency. (A) \log_{10} equivalent branching factor, b_{exh} and b_{any} . (B) Equivalent search depth, d_{exh} and d_{any} . (C) \log_{10} equivalent probability of success per draw, P_{exh} and P_{any} . In all graphs, the x -axis is the \log_{10} of the search space size. The final anytime best (*any*) is marked by “x” and exhaustive search (*exh*) is marked by “o.” The points correspond to Figs. 2 and 3, with all data sets pooled. Points with $x > 32$ correspond to the large SCOP cores; their exhaustive search values separate because m is larger than in other data sets. The y -axis metrics are defined in the text. In (C), “o” follows $y = -x$ and is truncated at $y = -10$ as uninformative.

It would be premature to conclude that the global optimum is known in any unproven case with certainty, or even with high probability. The point (32.6, 4.2) in Fig. 3A and Fig. 4 indicates that occasional extreme performance outliers may occur in realistic data (the heavy-fail phenomenon). Any approximate algorithm returns only the best results that could be found with limited search effort. Without a proof, it can never be guaranteed that the returned alignments are globally optimal.

Finally, the basic ideas above should apply to related divide-and-conquer algorithms. For example, both the generalization of this paper to gaps anywhere, and the divide-and-conquer method of Xu and Uberbacher (1996; Xu *et al.*, 1998) might be adaptable to an anytime control architecture.

APPENDIX A. PSEUDO-CODE

This Appendix gives pseudo-code for the control architecture described above.

A.1. Opportunistic mode pseudoprobability formulas

These formulas compute a crude heuristic approximation of the probability that a given hyper-rectangle contains a threading that will score better than the worst-scoring of the current best K candidates. Although probabilistic terminology is used for convenience, the resulting quantity is not really a probability in any meaningful sense of the word; it is just a search control heuristic.

A.1.1. Update. When a hyper-rectangle is split, the `update()` function is called once for each resulting child. `Lb-old` and `Q-ndx-old` are the lower bound and queue index of the parent; `Lb-new` and `Q-ndx-new` are the lower bound and queue index of the child. Arrays are initialized to zero.

TABLE 1. SCOP FIVE LARGEST SEQUENCES VS. FIVE LARGEST CORES: SEARCH RANKS AND TIMES

No.	Seq	Core	Seq Len	Core Segs	Size	Rank Lim	Rank Any	Max Δ rank	Sec Lim	Sec Any	Threadings per Sec Any
1	liphA2	llci	597	36	9.2d+42	104	4	1	74298	348	2.7d+40
2	lvnc	llci	609	36	4.0d+43	103	3	2	84809	375	1.1d+41
3	liphA2	lcxsA2	597	42	5.1d+43	108	8	1	27221	393	1.3d+41
4	lcxsA2	llci	625	36	2.6d+44	107	7	1	39013	486	5.3d+41
5	lvnc	lcxsA2	609	42	5.5d+44	135	35	25	47310	8360	6.6d+40
6	liphA2	loen	597	39	9.5d+45	104	4	3	49661	386	2.5d+43
7	lcxsA2	lcxsA2	625	42	1.1d+46	107	7	1	54124	419	2.6d+43
8	lvnc	loen	609	39	4.5d+46	107	7	1	66278	663	6.8d+43
9	2sblB1	llci	690	36	2.0d+47	107	7	3	38049	931	2.1d+44
10	lcxsA2	loen	625	39	3.3d+47	107	7	3	36084	761	4.3d+44
11	liphA2	4aahA	597	39	4.4d+49	109	9	4	32840	1878	2.3d+46
12	liphA2	ltsp	597	40	1.5d+50	111	11	5	36473	1850	8.1d+46
13	lvnc	4aahA	609	39	1.7d+50	107	7	4	32091	875	1.9d+47
14	2sblB1	lcxsA2	690	42	3.0d+50	107	7	1	59142	543	5.6d+47
15	2sblB1	loen	690	39	4.0d+50	116	16	10	27639	2686	1.5d+47
16	lvnc	ltsp	609	40	7.9d+50	113	13	4	42485	1820	4.3d+47
17	lbgw	llci	793	36	8.5d+50	129	29	1	83361	10535	8.1d+46
18	lcxsA2	4aahA	625	39	9.6d+50	109	9	1	29742	1343	7.2d+47
19	lcxsA2	ltsp	625	40	6.5d+51	103	3	2	61700	445	1.5d+49
20	2sblB1	4aahA	690	39	5.3d+53	107	7	3	55726	873	6.1d+50
21	lbgw	loen	793	39	3.2d+54	111	11	2	42168	3568	8.9d+50
22	2sblB1	ltsp	690	40	1.3d+55	111	11	3	80361	2475	5.2d+51
23	lbgw	lcxsA2	793	42	6.2d+55	103	3	2	62434	815	7.6d+52
24	lbgw	4aahA	793	39	1.9d+57	107	7	2	36534	2621	7.4d+53
25	lbgw	ltsp	793	40	2.1d+59	105	5	4	171377	1369	1.5d+56

The entries correspond to some search space sizes above 10^{40} in the figures. Seq and Core are identifiers in the SCOP (Murzin *et al.*, 1995) core domain database. Seq Len is the sequence length, Core Segs is the number of core secondary structure segments, and Size is the resulting search space size. Rank Lim is the total number of candidate alignments that were enumerated before search was abandoned (100 candidates beyond the final anytime best without improvement). Rank Any is the rank of the final anytime best in the list of candidate alignments. Max Δ rank is the maximum amount by which the rank of the current anytime best changed during the search. Sec Lim is the total number of seconds at which the search was abandoned. Sec Any is the total number of seconds at which the final anytime best candidate was found. Threadings per SecAny is the ratio of Size to Sec Any.

PARAMETER MAX-Z-DELTA 2.0;

PARAMETER MAX-DECAY 0.95;

GLOBAL VARIABLE Total-Counts[m + 1];

GLOBAL VARIABLE Update-Counts[m + 1];

GLOBAL VARIABLE Decay[m + 1];

GLOBAL VARIABLE Mean[m + 1];

GLOBAL VARIABLE Var[m + 1];

DEFINE update(Init, Lb-old, Q-ndx-old, Lb-new, Q-ndx-new) BEGIN

IF (Q-ndx-new = Q-ndx-old + 1) THEN BEGIN

Total-Counts[Q-ndx-old] \leftarrow Total-Counts[Q-ndx-old] + 1;

Delta \leftarrow Lb-new - Lb-old;

IF (0 < Var[Q-ndx-old])

THEN Z-Delta \leftarrow (Delta - Mean[Q-ndx-old]) / sqrt(Var[Q-ndx-old]);

ELSE Z-Delta \leftarrow 0;

WHEN (Z-Delta < MAX-Z-DELTA) OR (Var[Q-ndx-old] < 1) OR (Init = TRUE)

DO update-one(Q-ndx-old, Delta);

END

END

TABLE 2. SCOP FIVE LARGEST SEQUENCES VS. FIVE LARGEST CORES: SEARCH METRICS

No.	<i>LBEvals</i>	<i>LBEvals</i>	<i>B</i>	<i>B</i>	<i>B</i>	<i>D</i>	<i>D</i>	<i>D</i>	<i>P</i>	<i>P</i>	<i>P</i>
	<i>Lim</i>	<i>Any</i>	<i>Exh</i>	<i>Lim</i>	<i>Any</i>	<i>Exh</i>	<i>Lim</i>	<i>Any</i>	<i>Exh</i>	<i>Lim</i>	<i>Any</i>
1	197139	302	3.299	1.158	1.071	36.00	4.44	2.08	1.1d-43	5.1d-06	3.3d-03
2	194519	203	3.357	1.158	1.066	36.00	4.37	1.91	2.5d-44	5.1d-06	4.9d-03
3	224813	1832	2.831	1.136	1.081	42.00	5.14	3.14	2.0d-44	4.4d-06	5.5d-04
4	198574	751	3.434	1.159	1.083	36.00	4.29	2.33	3.9d-45	5.0d-06	1.3d-03
5	294197	61963	2.902	1.139	1.121	42.00	5.13	4.50	1.8d-45	3.4d-06	1.6d-05
6	214193	310	3.251	1.146	1.066	39.00	4.52	2.11	1.1d-46	4.7d-06	3.2d-03
7	236079	1447	2.992	1.136	1.078	42.00	4.90	2.88	9.2d-47	4.2d-06	6.9d-04
8	218064	1816	3.308	1.147	1.087	39.00	4.46	2.72	2.2d-47	4.6d-06	5.5d-04
9	193782	2045	3.720	1.158	1.096	36.00	4.02	2.52	5.1d-48	5.2d-06	4.9d-04
10	222733	2339	3.382	1.147	1.090	39.00	4.39	2.77	3.0d-48	4.5d-06	4.3d-04
11	229792	6813	3.571	1.147	1.103	39.00	4.21	3.01	2.3d-50	4.4d-06	1.5d-04
12	235039	11260	3.506	1.144	1.107	40.00	4.28	3.23	6.7d-51	4.3d-06	8.9d-05
13	221930	2307	3.625	1.147	1.090	39.00	4.15	2.61	5.9d-51	4.5d-06	4.3d-04
14	239789	933	3.326	1.137	1.073	42.00	4.48	2.47	3.3d-51	4.2d-06	1.1d-03
15	242576	19763	3.660	1.148	1.116	39.00	4.15	3.31	2.5d-51	4.1d-06	5.1d-05
16	223263	11192	3.570	1.143	1.107	40.00	4.20	3.18	1.3d-51	4.5d-06	8.9d-05
17	244355	45157	4.115	1.161	1.138	36.00	3.81	3.29	1.2d-51	4.1d-06	2.2d-05
18	224346	6442	3.696	1.147	1.103	39.00	4.09	2.91	1.0d-51	4.5d-06	1.6d-04
19	223020	217	3.652	1.143	1.060	40.00	4.13	1.80	1.5d-52	4.5d-06	4.6d-03
20	218943	571	3.965	1.147	1.073	39.00	3.88	2.00	1.9d-54	4.6d-06	1.8d-03
21	217685	10333	4.045	1.147	1.108	39.00	3.82	2.87	3.1d-55	4.6d-06	9.7d-05
22	236442	6907	3.966	1.144	1.101	40.00	3.90	2.79	7.7d-56	4.2d-06	1.4d-04
23	231206	236	3.775	1.136	1.058	42.00	4.04	1.79	1.6d-56	4.3d-06	4.2d-03
24	217909	2261	4.345	1.147	1.090	39.00	3.63	2.28	5.2d-58	4.6d-06	4.4d-04
25	222145	445	4.406	1.143	1.068	40.00	3.61	1.79	4.8d-60	4.5d-06	2.2d-03

The entries correspond to Table 1 according to the No. column. LBEvals is the number of lower bound evaluations. B is the branching factor b , D is the search depth d , and P is the probability of success per draw P , as defined in the text. Exh is exhaustive search, Lim is the point at which search was abandoned, and Any is the final anytime best. LBEvals Exh is not shown because it is equal to Size in Table 1. The notation $x\text{dy}$ means $x \times 10^y$.

TABLE 3. PROBLEM DIFFICULTY ACROSS DIFFERENT DATA SETS AND OBJECTIVE FUNCTIONS

<i>Obj Fcn</i>	<i>Data set</i>	<i>Mean rank</i>	<i>Mean max ΔR</i>	Anybest (all)		Anybest (proven)		Anytime proof		Batch proof	
				<i>Slope</i>	<i>Interc.</i>	<i>Slope</i>	<i>Interc.</i>	<i>Slope</i>	<i>Interc.</i>	<i>Slope</i>	<i>Interc.</i>
BL-93	SCOP	5.81	2.37	0.03	1.19	0.02	1.34	0.10	0.90	—	—
BL-93	Rost	1.20	1.31	0.09	0.06	0.09	0.06	0.11	0.14	—	—
BL-93	LS	1.24	1.33	0.09	0.11	0.09	0.11	0.12	0.12	0.13	-0.47
WMS-94	LS	3.22	1.73	0.11	-0.16	0.11	-0.15	0.15	-0.23	0.18	-0.62
S-93	LS	5.97	2.47	0.13	0.19	0.13	0.15	0.21	-0.16	0.24	-0.81

Obj Fcn is the objective function used: BL-93 is Bryant and Lawrence (1993), WMS-94 is White *et al.* (1994), and S-93 is Sippl (1993). These three objective functions have been characterized previously against the LS data set (Lathrop and Smith, 1996). Mean rank is the average rank of the anytime best, and Mean max ΔR is the average maximum change in the anytime best rank, across all trials. Anybest represents the time at which the anytime best was produced, Anytime proof represents the time at which the anytime best was proven to be globally optimal, and Batch proof is taken from Lathrop and Smith (1996) and is the time at which the batch branch-and-bound algorithm of Lathrop and Smith (1996) both produced and proved the global optimum. (all) indicates that all trials were included, and (proven) indicates only trials in which the anytime best was proven to be globally optimal were included. Slope and Interc. are, respectively, the slope and intercept of the best-fitting regression line to the plot of \log_{10} (elapsed seconds) vs. \log_{10} (search space size). SCOP, Rost, and LS correspond to Fig. 3A, B, and C, respectively.

```

DEFINE update-one(Q-ndx-old, Delta) BEGIN
Update-Counts [Q-ndx-old] ← Update-Counts [Q-ndx-old] + 1;
Decay [Q-ndx-old] ← min (MAX-DECAY,
    Update-Counts [Q-ndx-old] / (2 + Update-Counts [Q-ndx-old]));

```

```

Mean[Q-ndx-old] ← (Decay[Q-ndx-old] * Mean[Q-ndx-old])
  + ((1 - Decay[Q-ndx-old]) * Delta) ;
Var[Q-ndx-old] ← (Decay[Q-ndx-old] * Var[Q-ndx-old])
  + ((1 - Decay[Q-ndx-old]) * (Delta - Mean[Q-ndx-old])2)
  * ((max 2, Update-Counts[Q-ndx-old])
    / (max 1, Update-Counts[Q-ndx-old] - 1));
END

```

A.1.2. Log (pseudoprobability of better score). Log of the pseudo-probability that lower bound Lb at Q-ndx will eventually produce a score better than X. The function `left-normal-tail(X)` returns the log of the area under a normal curve to the left of X, and is based on a formula for the complementary error function which has a fractional error everywhere less than 1.2×10^{-7} (Press *et al.*, 1992).

```

DEFINE log-p-better(Lb, Q-ndx, X) BEGIN
IF (Lb > X) THEN RETURN(MINUS-INFINITY)
ELSE BEGIN
  M ← V ← 0;
  FOR ndx FROM Q-ndx TO MAX-Q-NDX DO BEGIN
    M ← M + Mean[ndx];
    V ← V + Var[ndx];
  END;
  IF (V = 0)
    THEN IF ((Lb + M) < X)
      THEN RETURN(0);
      ELSE RETURN(MINUS-INFINITY);
  ELSE BEGIN
    S ← sqrt(V);
    Z ← (X - (Lb + M)) / S;
    Left-Tail-Z ← left-normal-tail(Z);
    Z-Lb ← - M / S;
    Left-Tail-Z-Lb ← left-normal-tail(Z-Lb);
  END;
  RETURN(log(exp(Left-Tail-Z) - exp(Left-Tail-Z-Lb)));
END

```

A.2. Choose queue

```

PARAMETER MIN-INIT-SWEEPS 5;
PARAMETER K-BEST-DESIRED 1;
PARAMETER C-MAX-SPACE 100000;
GLOBAL VARIABLE Priority-Queue Q[m + 1];
GLOBAL VARIABLE Priority-Queue AQ[m + 1];

```

Iter is the total number of lower-bound splits performed so far. N-Queues (= $m + 1$) is the total number of primary queues Q (also the total number of annihilation queues AQ). The function `total-size()` returns the total number of hyper-rectangles in all the queues of its argument. The function `highest-occupied-queue()` returns its argument's highest-numbered queue holding any hyper-rectangles. The function `best-queue()` returns its argument's queue having the highest value of `log-p-better()`. The function `insert()` inserts a hyper-rectangle into its appropriate queue. The function `queue-pop()` removes and returns the top of its argument queue. The function `queue-empty()` returns TRUE if its argument queue is empty, else FALSE.

```

DEFINE choose-queue(Iter, N-Queues) BEGIN
IF (no current candidates) THEN
  RETURN(highest-occupied-queue(Q));

```

```

ELSE IF total-size(AQ) > 0 THEN
  RETURN(highest-occupied-queue(AQ));
ELSE IF total-size(Q) > C-MAX-SPACE THEN BEGIN
  insert(queue-pop(best-queue(Q)), AQ);
  RETURN(choose-queue(Iter, N-Queues));
END
ELSE IF ((number of current candidates < K-BEST-DESIRED)
  OR (Iter < N-Queues * MIN-INIT-SWEEPS))
  THEN BEGIN
    Sweep-Ndx ← mod(Iter, N-Queues);
    IF queue-empty(Q[Sweep-Ndx])
      THEN RETURN(best-queue(Q));
    ELSE RETURN(Q[Sweep-Ndx]);
  END;
ELSE RETURN(best-queue(Q));
END

```

ACKNOWLEDGMENTS

Temple Smith helped develop the original branch-and-bound algorithm. Bob Rogers helped implement the algorithm, and with Jadwiga Bienkowska kindly provided the SCOP and Rost (1997) data sets. Sandy Irani was an early advocate of anytime algorithms. Ljubomir Buturović, Raman Nambudripad, Srikar Rao, Chrysanthe Gaitatzes, Loredana Lo Conte, Barbara Bryant, Lisa Tucker-Kellog, and Sophia Zarakovich contributed greatly to core definition and construction. Comments from the blind reviewers improved the presentation. This paper describes research performed at the Information and Computer Science Department of the University of California, Irvine, sponsored by the National Science Foundation under Grant IRI-9624739. The program described in this paper is available electronically. An E-mail server is available by sending the word “help” to needle-request@darwin.bu.edu. See WWW <http://bmerc-www.bu.edu/needle/> or <http://www.bmerc.bu.edu/needle/>.

REFERENCES

- Akutsu, T., and Miyano, S. 1997. On the approximation of protein threading, 3–8. In Istrail, S., Karp, R., Lengauer, T., Pevzner, P., Shamir, R., and Waterman, M., eds., *Proceedings of the International Conference on Computational Molecular Biology*. ACM Press, New York.
- Bowie, J., and Eisenberg, D. 1993. Inverted protein structure prediction. *Curr. Opinion Struct. Biol.* 3, 437–444.
- Bryant, S.H., and Altschul, S.F. 1995. Statistics of sequence-structure threading. *Curr. Opinion Struct. Biol.* 5, 236–244.
- Bryant, S.H., and Lawrence, C.E. 1993. An empirical energy function for threading protein sequence through the folding motif. *Proteins: Struct. Funct. Genet.* 16, 92–112.
- Crawford, O.H. 1999. A fast, stochastic threading algorithm for proteins. *Bioinformatics.* 15, 66–71.
- Crippen, G.M. 1996. Failures of inverse folding and threading with gapped alignment. *Proteins* 26, 167–71.
- Fetrow, J.S., and Bryant, S.H. 1993. New programs for protein tertiary structure prediction. *Bio/Technology* 11, 479–484.
- Fischer, D., Elofsson, A., Rice, D., and Eisenberg, D. 1996. Assessing the performance of fold recognition methods by means of a comprehensive benchmark. In Hunter, L., and Klein, T., eds., *Proceedings of the Pacific Symposium on Biocomputing '96*. World Scientific Press, Singapore.
- Godzik, A., Kolinski, A., and Skolnick, J. 1992. Topology fingerprint approach to the inverse folding problem. *J. Mol. Biol.* 227, 227–238.
- Jernigan, R.L., and Bahar, I. 1996. Structure-derived potentials and protein simulations. *Curr. Opinion Struct. Biol.* 6, 195–209.
- Jones, D.T., and Thornton J.M. 1993. Protein fold recognition. *J. Computer-Aided Mol. Design* 7, 439–456.
- Jones, D.T., and Thornton, J.M. 1996. Potential energy functions for threading. *Curr. Opinion Struct. Biol.* 6, 210–216.
- Korf, R.E., and Reid, M. 1998. Complexity analysis of admissible heuristic search, 305–310. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98) and 10th Conference on Innovative Applications of Artificial Intelligence (IAAI'98)*. AAAI Press, Menlo Park, CA.
- Lathrop, R.H. 1994. The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Eng.* 7, 1059–1068.

- Lathrop, R.H. 1999. An anytime algorithm for gapped block protein threading with pair interactions, 238–249. In Istrail, S., Pevzner, P., and Waterman, M., eds., *Proceedings of the Third International Conference on Computational Molecular Biology*. ACM Press, New York.
- Lathrop, R.H., and Smith, T.F. 1996. Global optimum protein threading with gapped alignment and empirical pair score functions. *J. Mol. Biol.* 255, 641–665.
- Lathrop, R.H., Rogers, R.G., Jr., Smith, T.F., and White, J.V. 1998. A Bayes-optimal probability theory that unifies protein sequence-structure recognition and alignment. *Bull. Math. Biol.* 60, 1039–1071.
- Lawrence, C.E., Altschul, S.F., Boguski, M.S., Liu, J.S., Neuwald, A.F., and Wootton, J.C. 1993. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science* 262, 208–214.
- Lemer, C.M.-R., Rooman, M.J., and Wodak, S.J. 1995. Protein structure prediction by threading methods: Evaluation of current techniques. *Proteins: Struct. Funct. Genet.* 23, 337–355.
- Madej, T., Gibrat, J.-F., and Bryant, S.H. 1995. Threading a database of protein cores. *Proteins: Struct. Funct. Genet.* 23, 356–369.
- Moult, J., Pedersen, J.T., Judson, R., and Fidelis, K. 1995. A large-scale experiment to assess protein structure prediction methods. *Proteins: Struct. Funct. Genet.* 23, ii–iv.
- Murzin, A.G., Brenner, S.E., Hubbard, T., and Chothia, C. 1995. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.* 247, 536–540.
- Ouzounis, C., Sander, C., Scharf, M., and Schneider, R. 1993. Prediction of protein structure by evaluation of sequence-structure fitness. *J. Mol. Biol.* 232, 805–825.
- Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA.
- Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. 1992. *Numerical Recipes in C*, 2nd ed. Cambridge University Press, Cambridge.
- Rost, B., Sander, C., and Schneider, R. 1997. Protein fold recognition by prediction-based threading. *J. Mol. Biol.* 270, 471–480.
- Russell, R.B., and Barton, G.J. 1994. Structural features can be unconserved in proteins with similar folds. *J. Mol. Biol.* 244, 332–350.
- Sankof, D., and Kruskal, J.B., eds. 1983. *Time Warps, String Edits and Macromolecules*. Addison-Wesley, Reading, MA.
- Sippl, M.J. 1993. Boltzmann's principle, knowledge-based mean fields and protein folding. *J. Computer-Aided Mol. Design* 7, 473–501.
- Sippl, M.J. 1995. Knowledge-based potentials for proteins. *Curr. Opinion Struct. Biol.* 5, 229–235.
- Smith, T.F., Lo Conte, L., Bienkowska, J., Gaitatzes, C., Rogers, R.G., Jr., and Lathrop, R.H. 1997. Current limitations to protein threading approaches. *J. Comp. Biol.* 4, 217–225.
- Taylor, W.R., and Orengo, C.A. 1989. Protein structure alignment. *J. Mol. Biol.* 208, 1–22.
- Thomas, P.D., and Dill, K.A. 1996. Statistical potentials extracted from protein structures: How accurate are they? *J. Mol. Biol.* 257, 457–469.
- White, J., Muchnik, I., and Smith, T.F. 1994. Modeling protein cores with Markov random fields. *Math. Biosci.* 124: 149–179.
- Wodak, S.J., and Rooman, M.J. 1993. Generating and testing protein folds. *Curr. Opinion Struct. Biol.* 3:247–259.
- Xu, Y., and Uberbacher, C.E. 1996. A polynomial-time algorithm for a class of protein threading problems. *CABIOS* 12, 511–517.
- Xu, Y., Xu, D., and Uberbacher, C.E. 1998. An efficient computational method for globally optimal threading. *J. Comp. Biol.* 5, 597–614.

Address reprint requests to:

Richard H. Lathrop
Department of Information and Computer Science
University of California
Irvine, CA 92697

E-mail: rickl@uci.edu