An Anytime Algorithm for Gapped Block Protein Threading with Pair Interactions

Richard H. Lathrop
Department of Information and Computer Science,
University of California, Irvine, CA 92717-3425 USA
rickl@uci.edu

Abstract

This paper describes a novel anytime branch-and-bound 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. Using previously published data sets and the Bryant-Lawrence (1993) objective function, the algorithm found the true (proven) global optimum in less than five minutes in all search spaces size 10^{25} or smaller (sequences to 478 residues), and a putative (not guaranteed) optimum in less than five hours in all search spaces size 10^{60} or smaller (sequences to 793 residues, cores to 42 secondary structure segments). If the threading in the largest case were eventually proven to be globally optimal, then the corresponding search speed in the largest case studied would be the equivalent of 1.5×10^{56} threadings per second, 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 is given. The general approach should apply to other alignment methodologies and search methods that use a divide-and-conquer strategy.

Keywords:
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 3-D fold by assigning each sequence residue the 3-D coordinates of the aligned structure residue. For reviews see [2], [3], [6], [8], [9], [10], [15], [23], [26], while for cautionary notes see [5], [13], [16], [18], [21], [24], [25].

The protein sequence-structure alignment problem consists of a sequence, a structure (considered as drawn from a library of structures [14]), 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, pseudo-energy, 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 [7].

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 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 [4] was the fastest of five objective functions studied [13]. Speeds have been reported of 1.4×10^2 threadings/second for exhaustive search [4] and 6.8×10^{28} for branch-and-bound search [13], 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:

- **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.

- **Pair Interactions: Modelled vs. Not.** If interactions between pairs of sequence residues are not modelled then the problem is low-order polynomial using dynamic programming [22]. In contrast, if both alignment gaps and pair interactions between sequence residues are modelled, then restricted subproblems may be polynomial of varying degree [27, 28, 29] while the general pair interaction problem treated here is NP-hard [1, 12].

- **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.

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. The algorithmic strategy employed is to adapt a batch branch-and-bound threading algorithm [13] to include an anytime control structure. The result is a branch-and-bound 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 resources of time and space to run to completion. Other exact pair interaction algorithms for this task are [4, 13, 27, 28, 29].

1.2.1 Batch Branch and Bound Background

The method presented here is a modified version of a batch branch-and-bound threading algorithm [13]. 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 hyper-rectangle $[\mathbf{b}, \mathbf{d}]$, 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 hyper-rectangle 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 hyper-rectangle to a lower bound on the possible values of the objective function on any point in the hyper-rectangle. A splitting function partitions a hyper-rectangle into a small set of mutually disjoint and exhaustive smaller hyper-rectangles, 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 hyper-rectangles, sorted by the lower bound.

Initially the queue holds a single hyper-rectangle, $[\mathbf{1}, \tilde{\mathbf{n}}]$, that covers the entire search space. At each step, the hyper-rectangle 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 hyper-rectangle on the queue can possibly achieve a lower score. Otherwise it is partitioned using the splitting function, and the resulting smaller hyper-rectangles 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 An Anytime Search Method

In this paper, the original algorithm [13] is modified to use $m+1$ priority queues. Queue Q_k holds hyper-rectangles of vector subspace dimension $m-k$. The hyper-rectangles 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 Figure 1) according to the number of aligned core segments, i.e., in decreasing order of hyper-rectangle dimension. The idea is that, if the lower bound is not too loose, then hyper-rectangles that enclose favorable threadings will tend to sort to the front of the queues in Figure 1 and from there migrate to the more fully instantiated higher-numbered queues.

When a hyper-rectangle is split, at least one resulting hyper-rectangle is guaranteed to decrease in dimension and so will move to a higher-numbered queue (arrows in Figure 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 alignment from the top of the current queue; (2) splitting it and reinserting the children into queues as appropriate; (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. While space precludes details, 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_j. These yield a very crude estimate of the probability that the top hyper-rectangle in each queue contains an alignment that scores better than the current anytime best. The hyper-rectangle with the highest crude estimate is chosen and split.

2.1.1 Overall Control Architecture

The overall control architecture is a combination of sweeping and opportunistic modes. First the queues are initialized. Q_0 holds a single hyper-rectangle containing the entire search space, $[\mathbf{1}, \tilde{\mathbf{n}}]$, and all other queues are empty. Initially a limited number of sweeps (currently five) are done in order to produce some good answers quickly and to initialize the mean and standard deviation estimates. Thereafter the algorithm alternates between opportunistically operating on the most productive regions of the search space (currently 95% of the time) and periodic sweeps to produce the next candidate alignment (currently 5%). Each sweep produces a new alignment from Q_m.

The algorithm remembers its best candidate so far (called the "anytime best" below), and presents this as its best guess if it is stopped before it terminates normally. Otherwise, if allowed sufficient time and space, eventually it reaches a state in which the top hyper-rectangle of every queue has a lower bound that is equal to or greater than the actual score of the anytime best candidate found so far. This constitutes an implicit proof that the current anytime best is in fact a global optimum, because no other hyper-rectangle 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 anytime best is globally optimal, and then terminates normally.

2.1.2 An Improved Splitting Function

The original batch splitting function [13] partitioned hyper-rectangles according to a complicated function of partial probabilities over partial alignment scores. A simpler and more effective splitting function is:

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 remaining ties by splitting on the segment furthest from any fixed segment;
4. Break remaining ties by splitting on the segment closest to the middle of the structure.

2.2 Implementation and Hardware

The algorithm is implemented in Common Lisp. The results below were obtained on a 110 MHz SPARC-

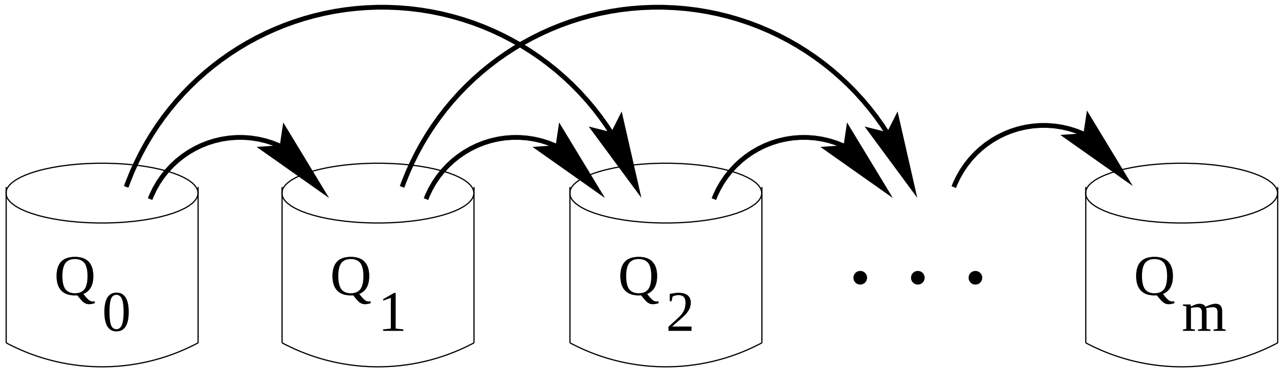


Figure 1: Schematic illustration of anytime queue cascade. Queue Q_0 holds completely uninstantiated alignments; Queue Q_m holds fully instantiated alignments; and intermediate queues Q_k hold partial alignments corresponding to hyper-rectangles of dimension $m-k$. Here, queue Q_k holds partial alignments 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).

station 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 & Lawrence (1993) [4]. Data sets were taken from Lathrop & Smith [13], Rost [20], and a SCOP [17] core domain database. The data set of Lathrop & Smith [13] 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 [20] 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 [17] core domain database the 5 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 & Smith [13] and Rost [20]. On the sequences and cores from SCOP [17] 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 & Smith [13] and Rost [20] is a true global optimum (proven), while many anytime bests in the SCOP largest cores data set are only putative (not guaranteed).

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 one 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 b^d leaf nodes. For exhaustive search, $b^d = S$ and $d = m$, so the branching factor for exhaustive search is $b_{exh} = S^{1/m} = \sqrt[m]{S}$. The equivalent branching factor for the algorithm [19] is $b_{alg} = L_{alg}^{1/m} = \sqrt[m]{L_{alg}}$.

2.4.2 Search Depth

Recent experimental results [11] 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 $d_{exh} = \log S/\log b_{exh} = m$, and $d_{alg} = \log L_{alg}/\log b_{exh}$.

2.4.3 Probability of Success

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

3 Results

Figure 2(A) 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 2(B) shows a histogram of the maximum Δrank observed. The Δrank 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 minutes). In Figure 3(A) an "L" marks the time at which search was abandoned iff the global optimum was not proven. In Figures 3(B) 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 five minutes 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 Figures 2 and 3 depend on search space size. Figure 4(A) shows the rank of the final anytime best, and Figure 4(B) shows the maximum Δrank observed. Figure 4(C) shows the total elapsed seconds/$m^2\tilde{n}^2$, i.e., the total time divided by the formal complexity of the lower bound calculation which is $\mathcal{O}(m^2\tilde{n}^2)$.

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

Tables 1 and 2 provide detailed analyses of the 5 largest SCOP cores against the 5 longest SCOP sequences. Note that in approximately half (13/25) of the cases shown, the final anytime best was obtained in 15 minutes or less total elapsed time.

4 Discussion

The anytime branch-and-bound 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. Because the 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 Figure 3(A), 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, Δranks, search times, branching factors, search depths, and probabilities per draw are comparable.

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 Figure 3(A) and Figure 4 indicates that occasional extreme performance outliers may occur in realistic data. Any approximate algorithm returns only the best results that could be found with limited search effort. Without a proof, one can never guarantee 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 [27, 28, 29], might be adaptable to an anytime control architecture.

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 Zarakhovich 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 email 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

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

[2] Bowie, J., Eisenberg, D. Inverted protein structure prediction. Current Opinion in Structural Biol. 3:437–444, 1993.

[3] Bryant, S.H., Altschul, S.F. Statistics of sequence-structure threading. Current Opinion in Structural Biol. 5:236–244, 1995.

[4] Bryant, S.H., Lawrence, C.E. An empirical energy function for threading protein sequence through the folding motif. Proteins: Structure, Function, and Genetics 16:92–112, 1993.

[5] Crippen, G.M. Failures of inverse folding and threading with gapped alignment. Proteins. 26:167-71, 1996.

[6] Fetrow, J.S., Bryant, S.H. New programs for protein tertiary structure prediction. Bio/Technology 11:479–484, 1993.

[7] Fischer, D., Elofsson, A., Rice, D., Eisenberg, D. Assessing the performance of fold recognition methods by means of a comprehensive benchmark. In *Proc. Pacific Symposium on Biocomputing'96,* ed. Hunter, L., Klein, T., World Scientific Press, Singapore.

[8] Jernigan, R.L., Bahar, I. Structure-derived potentials and protein simulations. Current Opinion in Structural Biol. 6:195–209, 1996.

[9] Jones, D. T., Thornton, J. M. Protein fold recognition. J. Computer-Aided Mol. Design. 7:439–456, 1993.

[10] Jones, D. T., Thornton, J. M. Potential energy functions for threading. Current Opinion in Structural Biol. 6:210–216, 1996.

[11] Korf, R.E., Reid, M. Complexity analysis of admissable heuristic search. In *Proc. 15th Natl. Conf. on Artificial Intelligence (AAAI'98) and 10th Conf. on Innovative Applications of Artificial Intelligence (IAAI'98),* AAAI Press, Menlo Park, CA, USA, 305–310, 1998.

[12] Lathrop, R.H. The protein threading problem with sequence amino acid interaction preferences is NP-complete. Protein Engng. 7:1059–1068, 1994.

[13] Lathrop, R.H., Smith, T.F. Global optimum protein threading with gapped alignment and empirical pair score functions. J. Mol. Biol. 255:641–665, 1996.

[14] Lathrop, R.H., Rogers Jr., R.G., Smith, T.F., White, J.V. A Bayes-optimal probability theory that unifies protein sequence-structure recognition and alignment. Bull. Math. Biol. 60:1039–1071, 1998.

[15] Lemer, C.M.-R., Rooman, M.J., Wodak, S.J. Protein structure prediction by threading methods: Evaluation of current techniques. Proteins: Structure, Function, and Genetics, 23:337–355, 1995.

[16] Moult, J., Pedersen, J.T., Judson, R., Fidelis, K. A large-scale experiment to assess protein structure prediction methods. Proteins: Structure, Function, and Genetics, 23:ii–iv, 1995.

[17] Murzin A.G., Brenner S.E., Hubbard T., Chothia C. SCOP: A structural classification of proteins database for the investigation of sequences and structures. J. Mol. Biol. 247:536–540, 1995.

[18] Ouzounis, C., Sander, C., Scharf, M., Schneider, R. Prediction of protein structure by evaluation of sequence-structure fitness. J. Mol. Biol. 232:805–825, 1993.

[19] Pearl, J. *Heuristics: Intelligent search strategies for computer problem solving.* Addison-Wesley, Reading, Massachusetts, 1984.

[20] Rost, B., Sander, C., Schneider, R. Protein fold recognition by prediction-based threading. J. Mol. Biol., 270:471–480, 1997.

[21] Russell, R.B., Barton, G.J. Structural features can be unconserved in proteins with similar folds. J. Mol. Biol. 244:332–350, 1994.

[22] Sankof, D., Kruskal, J.B., eds. Time warps, string edits and macromolecules. Addison-Wesley, Reading, MA, USA, 1983.

[23] Sippl, M.J. Knowledge-based potentials for proteins. Current Opinion in Structural Biol. 5:229–235, 1995.

[24] Smith, T.F., Lo Conte, L., Bienkowska, J. Gaitatzes, C., Rogers Jr., R.G., Lathrop, R.H. Current limitations to protein threading approaches. J. Comp. Biol., 4:217–225, 1997.

[25] Thomas, P.D., Dill, K.A. Statistical potentials extracted from protein structures: How accurate are they? J. Mol. Biol. 257:457–469, 1996.

[26] Wodak, S. J., Rooman, M. J. Generating and testing protein folds. Current Opinion in Structural Biol. 3:247–259, 1993.

[27] Xu, Y., Uberbacher, C.E. A polynomial-time algorithm for a class of protein threading problems. CABIOS. 12:511–517, 1996.

[28] Xu, Y., Xu, D., Uberbacher, C.E. A new method for modeling and solving the protein fold recognition problem. pp. 285–292 in *Proc. Intl. Conf. on Computational Molecular Biology,* (ed. Istrail, S., Karp, R., Lengauer, T., Pevzner, P., Shamir, R., Waterman, M.), ACM Press, New York, 1998.

[29] Xu, Y., Xu, D., Uberbacher, C.E. An efficient computational method for globally optimal threading. J. Comp. Biol., 5:597–614, 1998.

Figure 2: Final Anytime Best Rank and Maximum ΔRank. (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 & Smith [13], horizontal stripes is Rost [20], and white is the 5 largest cores from the SCOP domain database [17]. For Lathrop & Smith [13] and Rost [20] the final anytime best was proven to be the global optimum; for SCOP [17] search was abandoned after 100 candidates were enumerated without improvement.

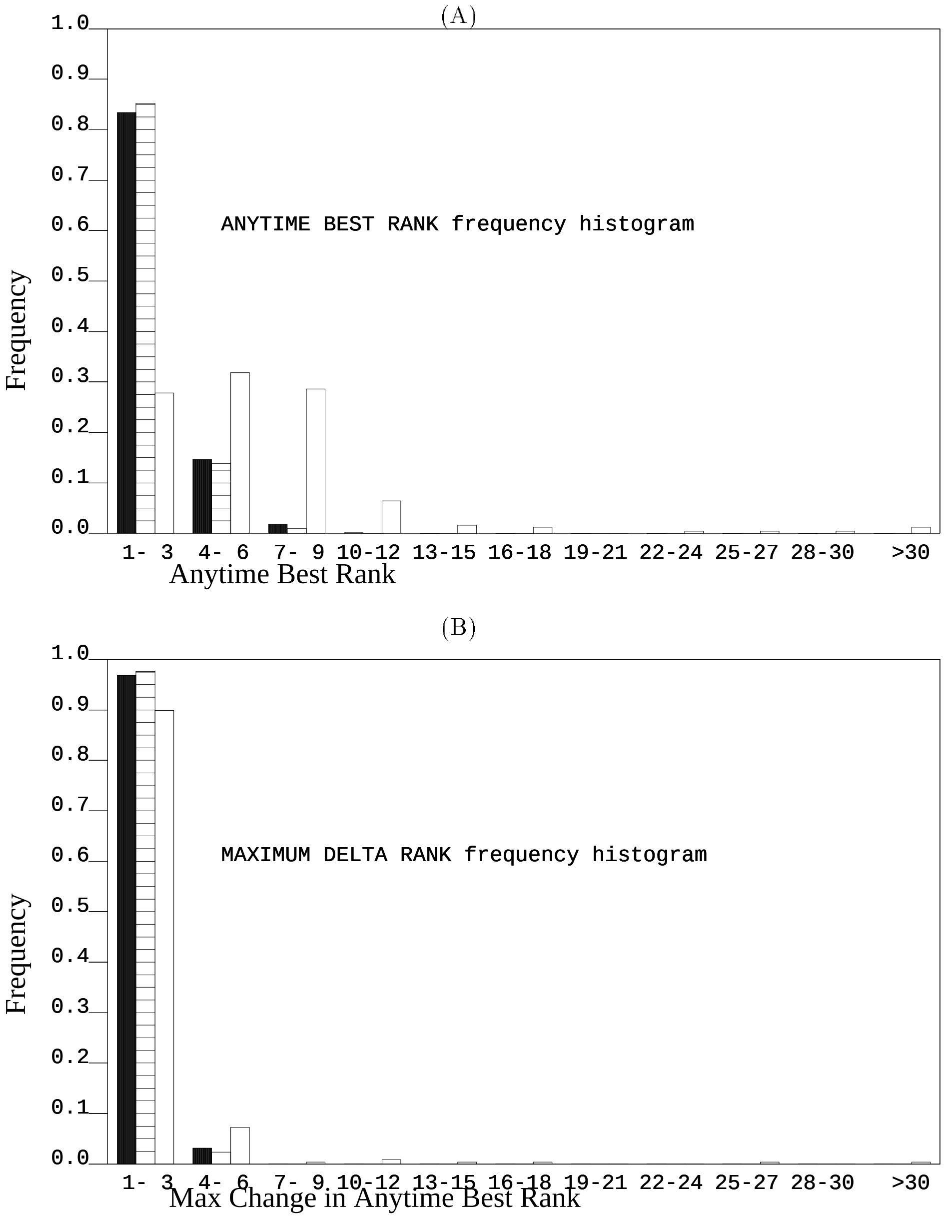


Figure 3: $\log_{10}$ Total Seconds to Final Anytime Best. (A) The 5 largest cores from the SCOP data set [17]. (B) The data set of Rost [20]. (C) The data set of Lathrop & Smith [13]. In all graphs the x-axis is the $\log_{10}$ of the search space size. The dashed line corresponds to 5 minutes. The runs correspond to Figure 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.

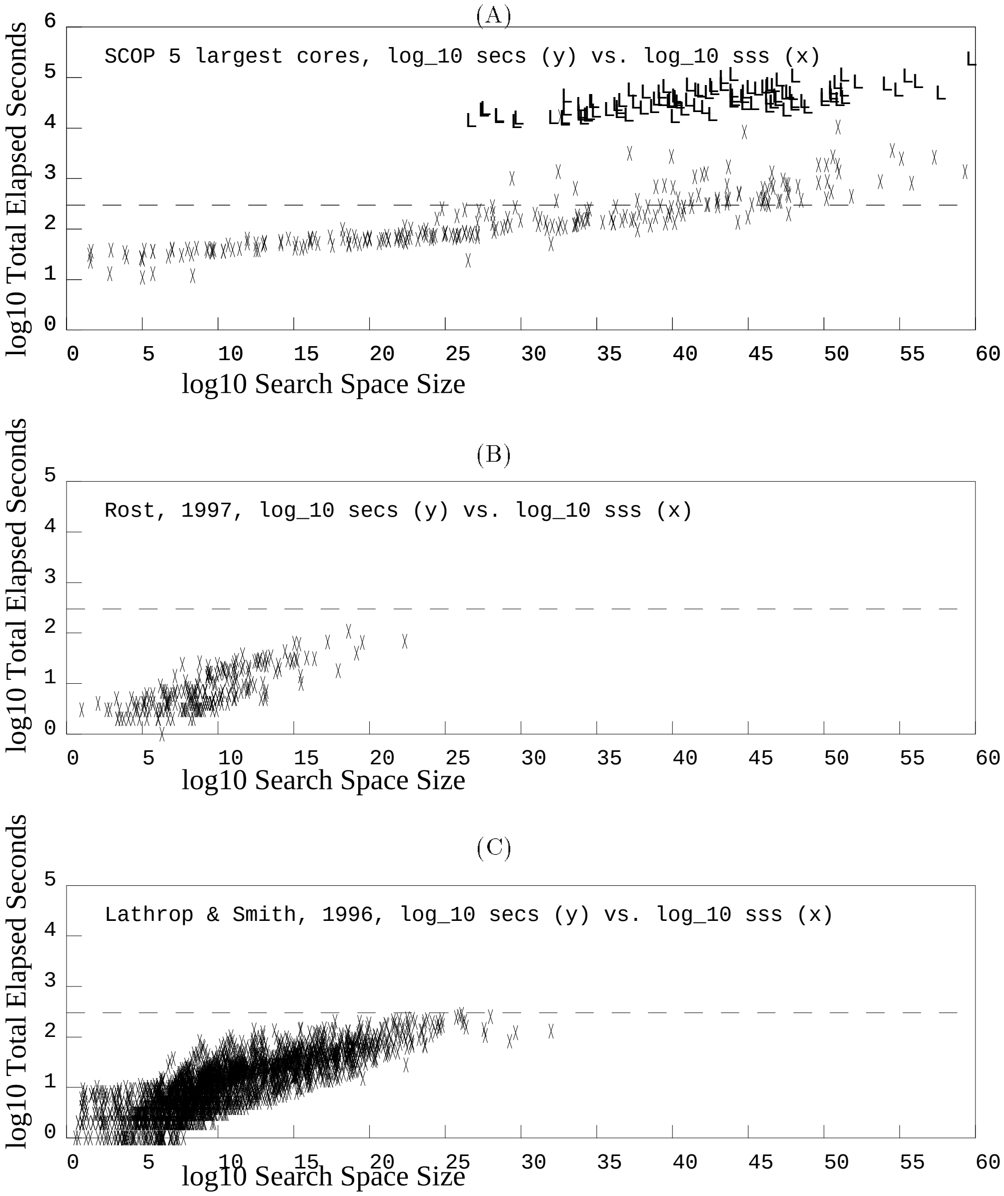


Figure 4: Search Space Size Dependencies. (A) Rank of Final Anytime Best; the points correspond to Figure 2A with all data sets pooled. (B) Maximum ΔRank; the points correspond to Figure 2B with all data sets pooled. (C) $\log_{10}$ (Total Seconds to Final Anytime Best / $(\tilde{n}^2 m^2)$); the points correspond to Figure 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 Figure 3(A). By coincidence, its maximum ΔRank was 100, numerically equal to the ΔRank cut-off for abandoning the search.

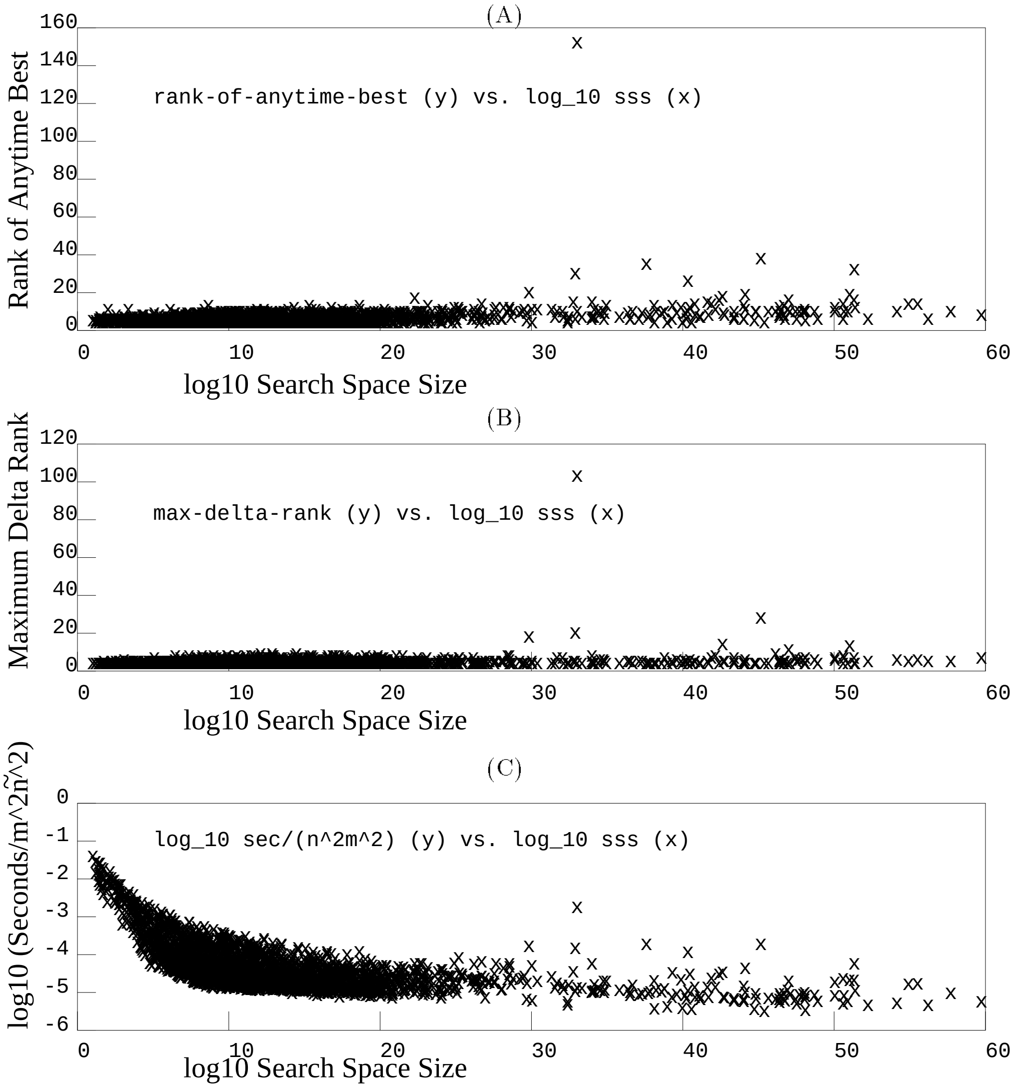
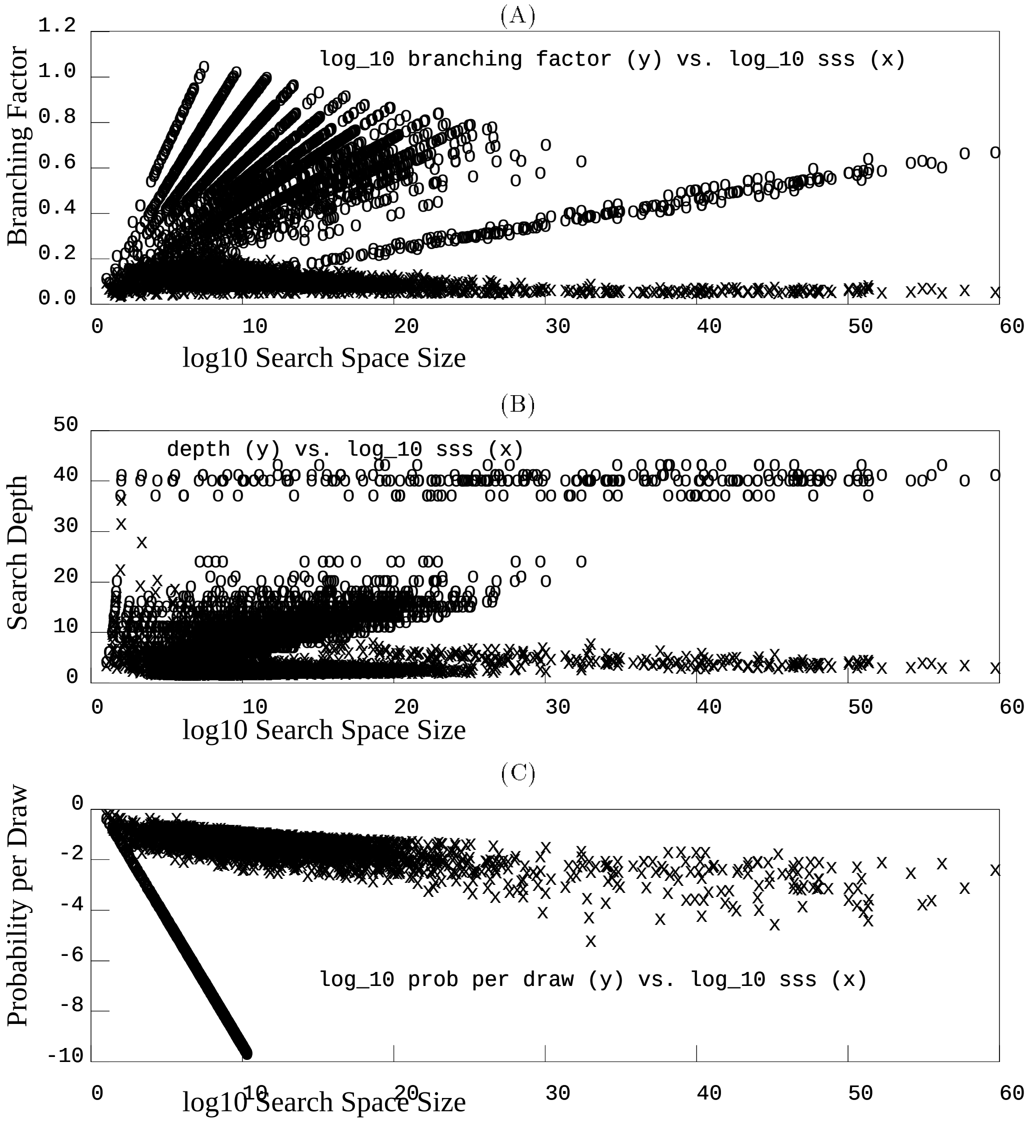


Figure 5: Implementation-Independent Metrics of Search Efficiency. (A) $\log_{10}$ Equivalent Branching Factor, b_{exh} and b_{any}. (B) Equivalent Search Depth, d_{exh} and d_{any}. (C) $\log_{10}$ Equivalent Probability of Success per Draw, P_{exh} and P_{any}. In all graphs, the x-axis is the $\log_{10}$ 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 Figures 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.



No.	Seq	Core	Seq Len	Core Segs	Size	Rank Lim	Rank Any	Max ΔRank	Sec Lim	Sec Any	Threadings per SecAny
1	1iphA2	1lci	597	36	9.2d+42	104	4	1	74298	348	2.7d+40
2	1vnc	1lci	609	36	4.0d+43	103	3	2	84809	375	1.1d+41
3	1iphA2	1cxsA2	597	42	5.1d+43	108	8	1	27221	393	1.3d+41
4	1cxsA2	1lci	625	36	2.6d+44	107	7	1	39013	486	5.3d+41
5	1vnc	1cxsA2	609	42	5.5d+44	135	35	25	47310	8360	6.6d+40
6	1iphA2	1oen	597	39	9.5d+45	104	4	3	49661	386	2.5d+43
7	1cxsA2	1cxsA2	625	42	1.1d+46	107	7	1	54124	419	2.6d+43
8	1vnc	1oen	609	39	4.5d+46	107	7	1	66278	663	6.8d+43
9	2sblB1	1lci	690	36	2.0d+47	107	7	3	38049	931	2.1d+44
10	1cxsA2	1oen	625	39	3.3d+47	107	7	3	36084	761	4.3d+44
11	1iphA2	4aahA	597	39	4.4d+49	109	9	4	32840	1878	2.3d+46
12	1iphA2	1tsp	597	40	1.5d+50	111	11	5	36473	1850	8.1d+46
13	1vnc	4aahA	609	39	1.7d+50	107	7	4	32091	875	1.9d+47
14	2sblB1	1cxsA2	690	42	3.0d+50	107	7	1	59142	543	5.6d+47
15	2sblB1	1oen	690	39	4.0d+50	116	16	10	27639	2686	1.5d+47
16	1vnc	1tsp	609	40	7.9d+50	113	13	4	42485	1820	4.3d+47
17	1bgw	1lci	793	36	8.5d+50	129	29	1	83361	10535	8.1d+46
18	1cxsA2	4aahA	625	39	9.6d+50	109	9	1	29742	1343	7.2d+47
19	1cxsA2	1tsp	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	1bgw	1oen	793	39	3.2d+54	111	11	2	42168	3568	8.9d+50
22	2sblB1	1tsp	690	40	1.3d+55	111	11	3	80361	2475	5.2d+51
23	1bgw	1cxsA2	793	42	6.2d+55	103	3	2	62434	815	7.6d+52
24	1bgw	4aahA	793	39	1.9d+57	107	7	2	36534	2621	7.4d+53
25	1bgw	1tsp	793	40	2.1d+59	105	5	4	171377	1369	1.5d+56

Table 1: SCOP 5 largest sequences vs. 5 largest cores — Search Ranks and Times. The entries correspond to some search space sizes above 10^{40} in the Figures. Seq and Core are identifiers in the SCOP [17] 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 ΔRank 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.

No.	LBEvals Lim	LBEvals Any	B Exh	B Lim	B Any	D Exh	D Lim	D Any	P Exh	P Lim	P Any
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

Table 2: SCOP 5 largest sequences vs. 5 largest cores — Search Metrics. 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.