A Multi-Queue Branch-and-Bound Algorithm for Anytime Optimal Search with Biological Applications

Richard H. Lathrop rickl@uci.edu
Anton Sazhin sazhin@ics.uci.edu
Ye Sun ysun@uci.edu
Nick Steffen nsteffen@ics.uci.edu
Sandra S. Irani irani@ics.uci.edu
Information and Computer Science, U. of California, Irvine, CA 92697-3425, U.S.A.

Abstract

Many practical biological problems involve an intractable (NP-hard) search through a large space of possibilities. This paper describes preliminary results from a multi-queue variant of branch-and-bound search that combines anytime and optimal search behavior. The algorithm applies to problems whose solutions may be described by an N-dimensional vector. It produces an approximate solution quickly, then iteratively improves the result over time until a global optimum is produced. A global optimum may be produced before producing its proof of global optimality. Local minima are never revisited. We describe preliminary applications to *ab initio* protein backbone prediction, small drug-like molecule conformations, and protein-DNA binding motif discovery. The results are encouraging, although still quite preliminary.

Keywords: heuristic search, branch-and-bound, protein folding, HP lattice, conformational search, DNA sequence motifs

1 Introduction

Search through large NP-hard search spaces is ubiquitous in biological applications, and a great many search algorithms have been proposed and explored (see Kanal & Kumar [8] and many others). Though other categorizations are useful as well, for purposes of this paper the algorithmic categories considered are based on the properties of optimal vs. approximate, complete vs. incomplete, anytime vs. batch, and irredundant vs. repetitive. An optimal algorithm proves that its solution is indeed the global optimum. An approximate algorithm produces a solution but no proof of global optimality; its optimality, if any, is unknown. A complete algorithm is guaranteed to find a solution if one exists, while an incomplete algorithm may fail to find any solution even if one is present in the search space. An anytime algorithm may be stopped any time after a short initialization and a usable answer obtained; it then iteratively improves that answer if allowed more time. A batch algorithm, if stopped early, produces no answer; its answer is produced only at the end of the run, when it terminates. An irredundant algorithm produces any given local minima no more than once. A repetitive algorithm may produce the same local minima repeatedly.

The algorithm in this paper is complete, optimal, anytime, and irredundant, and belongs to the family of search methods based on an admissable heuristic. It reduces to both depth-first branch-and-bound and A* search as special cases. It applies to problems whose solutions are an N-dimensional vector. It seeks to produce a good solution quickly, a global optima soon thereafter, and the proof of optimality eventually if allowed sufficient resources. This paper first describes the domain-independent search components and the domain-dependent interface. Then it describes briefly a number of subsequent applications of the basic algorithm. These include *ab initio* protein backbone prediction, small drug-like molecule conformational search, and protein-DNA binding motif discovery.

2 Methods

This section first describes the search algorithm components. Then it briefly describes several novel biological applications, including *ab initio* protein backbone prediction, small drug-like molecule conformational search, and protein-DNA binding motif discovery.

The main idea of the search algorithm is to maintain an explicit representation of the entire search space as a list of vector sub-spaces (axis-parallel hyper-rectangles). These partition the search space. The algorithm executes generalized branch-and-bound search using the hyper-rectangles to maintain state: splitting them into smaller hyper-rectangles (branch), computing a lower-bound on possible scores (bound), and discarding those whose lower bound is provably bad (prune). Hyper-rectangles are sorted by vector sub-space dimension into a series of priority queues, with each queue containing hyper-rectangles of the same dimensionality sorted by lower bound. If the lower-bound is a good heuristic, then hyper-rectangles that enclose favorable solutions will tend to sort to the front of the queues, and their descendants will migrate quickly to lower dimension queues. Sweeping across the queues from high to low dimensionality yields a complete solution rapidly (assuming all children are feasible). The algorithm also estimates the probability distributions over lower bound increases from queue to queue, and then estimates the queue likeliest to yield a better solution than the current best.

2.1 Search Space Representation

The algorithm applies to problems whose solutions may be represented by an N-dimensional vector, where N is fixed in advance. Thus a hyper-rectangle in vector space corresponds to a subset of the search space. A hyper-rectangle is fixed uniquely by its opposite corners, represented compactly as an ordered pair of N-dimensional vectors $[\vec{b}, \vec{d}]$. It contains all vectors $\vec{v}$ such that $b_i \leq v_i \leq d_i$, for $1 \leq i \leq N$. Its dimensionality is the number of coordinates i such that $b_i < d_i$ for integer-valued vectors, or $b_i + \epsilon < d_i$ for real-valued vectors. The entire search space may be represented by a single hyper-rectangle whose two corners enclose all legal solutions.

Search proceeds by choosing a hyper-rectangle and splitting it into mutually disjoint and exhaustive subsets, at least one of which has dimensionality less than the original hyper-rectangle. In the simplest case, a hyper-rectangle $[\vec{b}, \vec{d}]$ is split by choosing a split coordinate i and split value w such that $b_i \leq w \leq d_i$. The interval $[b_i, d_i]$ is divided into three sub-intervals. For integer-valued vectors these are the points (1) less than the split-point (BOT), $[b_i, w-1]$; (2) equal to the split-point (MID), $[w, w]$; and (3) greater than the split-point (TOP), $[w+1, d_i]$. For real-valued vectors these are (1) BOT, $[b_i, w-\epsilon/2]$; (2) MID, $[w-\epsilon/2, w+\epsilon/2]$; and (3) TOP, $[w+\epsilon/2, d_i]$. For space efficiency, represent only the most recent split point information plus a pointer to the parent, a few bytes of storage.

The multi-queue data structure consist of $N+1$ priority queues. Queue Q_k holds hyper-rectangles of dimension $N-k$. The hyper-rectangles in each queue are sorted by their queue-key, usually the lower-bound. The queues are initialized by a list of hyper-rectangles which cover the search space. Thereafter iteratively a queue is chosen, its top hyper-rectangle removed and split, and lower bounds calculated for the children. Children are pruned if they are infeasible or their lower bound exceeds the best solution so far. Surviving children of dimension k are inserted into queue Q_{N-k}.

2.2 Search Control Logic

Different operating modes concern which queue to select next; its top element is removed and split. If we always select the queue with the lowest lower bound we obtain A* search, while if we always select the highest-numbered occupied queue we obtain depth-first branch-and-bound search. However, multiple queues permit more fine-grained search control logic. Here we describe the two modes we have found most useful, sweep mode and probabilistic mode; other modes can be constructed easily from the basic data structures. In practice, we have found the best combination usually involves a few initial sweeps (default 10 sweeps) to generate some initial good solutions and initialize the

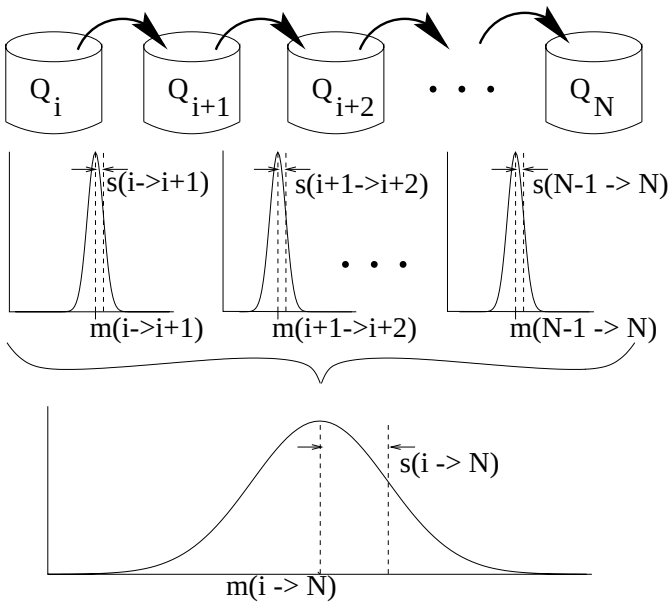

Figure 1: Schematic illustration of the distribution of lower bound increases. Top: Hyper-rectangles split and their children move to higher-numbered queues (arrows). Middle: Distribution of individual lower bound increases at each queue. Bottom: Distribution of cumulative lower bound increases. "m(j→k)" is the mean increase from Q_j to Q_k, and "s(j→k)" is the standard deviation.

probably estimates, followed by operating principally in probabilistic mode (default 90% of search effort) alternating with occasional sweeps (default 10% of search effort).

When a hyper-rectangle is split, at least one resulting hyper-rectangle (namely, MID) is guaranteed to decrease in dimension (provided it is feasible). It is always possible to "sweep" across the queues by beginning at the lowest-numbered occupied queue and at each step: (1) popping the top hyper-rectangle of the current queue; (2) splitting it and reinserting the children into queues as described above; (3) advancing to the next-highest-numbered occupied queue. One such sweep can be done in at most $3N$ lower bound evaluations, and provided all children are feasible it is guaranteed to produce a fully instantiated solution from Q_N at the end of the sweep (unless the children are pruned).

Probabilistic mode attempts to estimate probabilistically which queue's top hyper-rectangle is most likely to yield a descendant that ultimately scores better than the current best solution. The algorithm records the lower bound increases when a hyper-rectangle was split at queue Q_i and a child resulted at queue Q_{i+1}. From these it estimates the distribution of lower bound increases from each Q_i to Q_{i+1} as a decaying average of the mean and standard deviation. Since means and variances sum, this yields an estimate of the overall distribution from any given queue i to queue N. See Fig. 1.

From this we estimate the probability that a given hyper-rectangle at queue i will complete to a solution better than the current best. See Fig. 2. A normal distribution approximation is used, based on a fast error function approximation due to Chebyshev with a fractional error everywhere less than 1.2×10^{-7} [14]. We select the queue whose top hyper-rectangle has the highest estimated probability of bettering the current best solution. This not a true probability, but only a search control heuristic.

The pseudo-code given in Fig. 3 is slightly simplified for clarity relative to the actual running code. Procedure `Update` is called after each split. `Old-q` is the old queue index, `Old-lb` is the old lower bound, and `New-lb` is the new lower bound after the split. MAX-Z-DELTA (default 3.0), MIN-INIT-COUNTS (default 10), and MAX-DECAY (default 0.98) are parameters. Procedure `Log-p-better` in Fig. 3 returns the log of the probability that lower bound `Lb` at queue `Q-ndx` eventually will score better than `Current-best`. The function `log-one-tail-z(Z)` returns the log of the integrated tail of the normal distribution beyond `Z`.

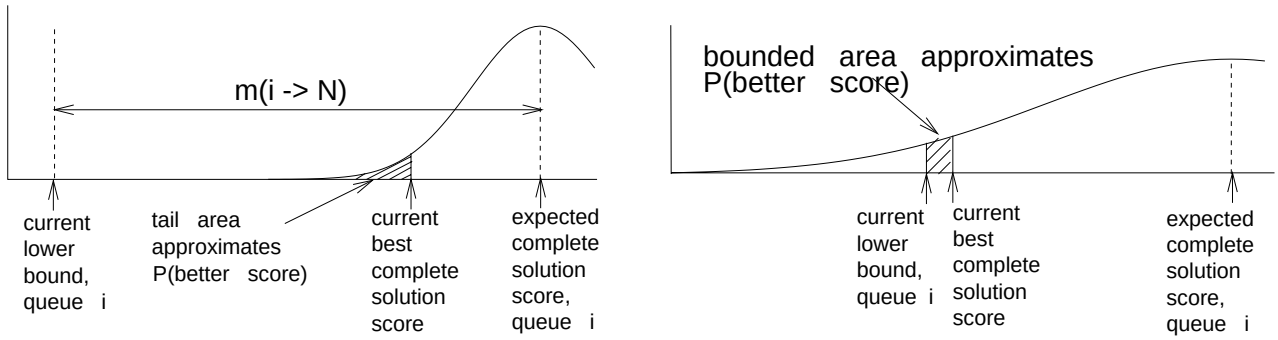

Figure 2: Probability of better score than current best. (LEFT) Lower bound far below current best score. (RIGHT) Lower bound very near current best score.

```
PROCEDURE Update(Old-q, Old-lb, New-lb)
BEGIN
Total-Counts[Old-q] ←
 Total-Counts[Old-q] + 1;
Delta-lb ← New-lb - Old-lb;
IF (0 < Var[Old-q])
 THEN Z-Delta-lb ←
  (Delta-lb - Mean[Old-q])
   / sqrt(Var[Old-q]);
 ELSE Z-Delta-lb ← 0;
IF (Z-Delta-lb < MAX-Z-DELTA)
  OR (Counts[Old-q] < MIN-INIT-COUNTS)
 DO BEGIN
 Var[Old-q] ←
  Decay[Old-q] * Var[Old-q]
   + (1 - Decay[Old-q])
    * (Delta - Mean[Old-q])^2;
 Mean[Old-q] ←
  Decay[Old-q] * Mean[Old-q]
   + (1 - Decay[Old-q]) * Delta;
 Decay[Old-q] ←
  min(MAX-DECAY,
   (1 + Counts[Old-q])
    / (2 + Counts[Old-q]));
 Counts[Old-q] ← Counts[Old-q] + 1;
 END
Sum-Means[0] ← Mean[0];
Sum-Vars[0] ← Var[0];
FOR I FROM 1 TO (NQUEUES - 1) DO BEGIN
 Sum-Means[I] ← Sum-Means[I-1] +
Mean[I];
 Sum-Vars[I] ← Sum-Vars[I-1] + Var[I];
 END
END
```

```
PROCEDURE Log-p-better
  (Lb, Q-ndx, Current-best) BEGIN
Mean ← Sum-Means[Q-Ndx];
Var ← Sum-Vars[Q-Ndx];
IF (Var = 0)
 THEN IF ((Lb + Mean) < Current-best)
  THEN RETURN(0.0);
  ELSE RETURN(-∞);
Sdev ← sqrt(Var);
Z ← ((Current-best - Lb - Mean) / Sdev);
IF ( Z > 0)
 THEN Log-P-Below-Z ←
  log( 1 - exp(log-one-tail-z(Z)));
 ELSE Log-P-Below-Z ← log-one-tail-z(z);
Z-Lb ← ( - Mean) / SDev;
Log-P-Below-Z-Lb ← log-one-tail-z(Z-Lb);
IF ( Z-Lb ≥ Z)
 THEN RETURN(MINUS-INFINITY)
 ELSE RETURN
  (log(exp(log-p-below-z)
     - exp(log-p-below-z-lb)));
END
```

Figure 3: Pseudocode. (LEFT) Probability update. (RIGHT) Probability of better score.

2.3 Protein Threading

The algorithm described here was developed originally to search large NP-hard search spaces in the domain of protein structure prediction by protein threading [10, 11]. Using previously published data sets and the Bryant-Lawrence objective function [2], 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). The threading in the largest case studied (search space size 2.1×10^{59}) was found in 23 minutes and proven optimal in 15 days. its search speed was the equivalent of 1.5×10^{56} threadings per second, a speed-up exceeding 10^{25} over previously published batch heuristic search speeds, and exceeding 10^{50} over previously published exhaustive search speeds.

2.4 *Ab Initio* Protein Backbone Prediction

The goal of this application is to predict 3D protein Cα (backbone) conformation based on the amino acid sequence information only. Input is a sequence which consists of m amino acid residues. The search seeks to find a legal 3D-space conformation of the sequence that has the minimum possible total energy. To simplify the discussion we refer to the sequence of m amino acid residues as a chain of m nodes; a conformation corresponds to assigning each node in the chain to a point in 3D space.

There are two different types of space models: discrete and continuous. In the discrete case a conformation is legal if the chain of nodes is fitted into a self-avoiding, gap-free path through vertices of the given 3D lattice. Lattice notation and background are given by Kittel [9]. More sophisticated lattices describe protein structure with higher fidelity but take much more computational effort [6]. In the continuous case a conformation is legal if every pair of nodes satisfies certain distance and handedness constraints learned from a database of known protein 3D structures.

The problem requires fixing three degrees of freedom for each node, corresponding to its (x, y, z) coordinates. Subspaces in the search space correspond to the Cartesian product of 3D volumes that contain each node. The 3D volume containing each node is represented by a polyhedron, as follows. Choose in advance a set of V basis vectors. The current set is the union of those corresponding to the cube and the octahedron (a total of seven basis vectors). Each polyhedron is defined by a set of planes, two planes perpendicular to each basis vector. The node can occupy any legal point inside the polyhedron. This representation allows fast splitting and intersection of spatial volumes.

The objective function to minimize is the total energy of the conformation, that is, the sum of all pairwise interaction energies between nodes. The interaction energies correspond to distance-dependent empirical, knowledge-based, or contact potentials [2]; these are learned from a database of known protein 3D structures, and fixed in advance. We use a matching technique to estimate a lower bound of some subset of conformations. The node set of the graph will have one node for each node in the chain. For each such node i, we associate an integer value $l(i)$ which will be the number of neighboring locations of that node type. These are learned from a database of known protein 3D structures. Put an undirected edge (i, j) into the graph for each pair of atoms i and j such that there is a possible location of i which is within neighbor distance of a possible location for j. Each edge (i, j) has a weight $c(i, j)$ which is the cost (energy) incurred if i and j are neighbors. Finally we solve the generalized matching problem in a bipartite graph, using a minimum cost flow algorithm [3, 7].

2.5 Conformations of Small Drug-Like Molecules

The goal of this application is low-energy conformation search of small drug-like molecules. Finding low energy molecular conformations is critical in rational drug design. We combine the DREIDING force field [13] and an implicit solvent model [12] to simulate molecules in solution. We have used the multi-queue approach to search for global minima of small molecules of size less than one hundred

atoms. We find this algorithm can effectively prune the very large search space, quickly get good approximate minima, iteratively improve the result, and eventually find global minima if sufficient time and space are allowed.

In the internal coordinate system, an atom has three coordinates: bond length, bond bending angle, and bond torsion angle. Each describes the location or relative rotation of one of the atoms in relation to the other atoms participating in the bonds. Thus, a molecular conformation can be represented as a vector of all the internal coordinates, and a set of conformations represented as a hyper-rectangle whose corners are vectors giving upper and lower bounds on their values.

The strategy we choose to split a set of conformations is to attempt to get a low energy conformation as early as possible. Because bond bending angle variation has more impact on the energy than torsion angle, for each atom, we split the bond bending angle first. We do a fine-grained discrete scan (currently in increments of 10 degrees) of the local energy variation about each internal coordinate before splitting, and choose the value that minimizes the local energetics.

The objective function to minimize is the atomic force field as modified by the implicit solvation model. The lower bounds for bonded interactions are easy to obtain based on minimal values of their local energetics during a scan across the allowed values. The lower bound for non-local interactions, e.g., van der Waals forces, is hard to estimate. We use several methods to obtain these: (1) incorporate 1-4 interaction into torsion interaction; (2) convert to global Cartesian coordinates for fixed atoms; (3) for other interactions, use the graph-based bipartite matching algorithm described above.

2.6 Protein-DNA Binding Motif Discovery

The sequence motif discovery problem consists of identifying common degenerate subsequences, such as binding sites and regulatory regions, in sets of biological sequences (RNA, DNA, or proteins). This application is implemented in C++.

A probability or weight matrix represents motifs. It contains one number for each symbol in the sequence alphabet across each position in the motif, giving probabilities or weights associated with each symbol occurrence at each motif position. Most of the currently used evaluation functions are based on the number and distribution of matches that the probability or weight matrix achieves on the sequences, and these may be bounded based on the upper and lower bounds on the probabilities or weights at each coordinate. This in turn would bound the objective function. As currently implemented, we use an information theoretic evaluation function to estimate the quality of any partial solution.

3 Results

This section presents results illustrating the algorithm finding low-energy molecular conformations, showing that the growth in run-time to find the best solution is slower than that to find the proof of global optimality, and demonstrating the anytime behavior by which good results are returned quickly and then improved rapidly as more time is allowed.

3.1 Illustrative Molecular Conformations

Fig. 4 shows two examples of molecular conformations found by the algorithm. On the left is an HP sequence of 120 residues folded on a cubic lattice under the HP force field. The sequence used is of a special form, $(\mathrm{PHP})^n$, built as the concatenation of n letter triplets "PHP." From symmetry arguments, its global optimum conformation is known to be a long helix with four H residues and eight P residues at each turn. While this doubtless leads to a somewhat simpler and faster search for the global optimum than in normal sequences, it is nonetheless a difficult search problem in its own right: the search space size for the sequence shown is approximately 2.4×10^{78}. On the right is an

HIV integrase inhibitor, L-chicoric acid, folded in off-lattice (continuous) 3D space under the Dreiding atomic force field with implicit solvent. The conformation is energetically favorable but no associated proof of optimality was obtained so its optimality is unknown. The search space size is approximately 3.7×10^{31}.

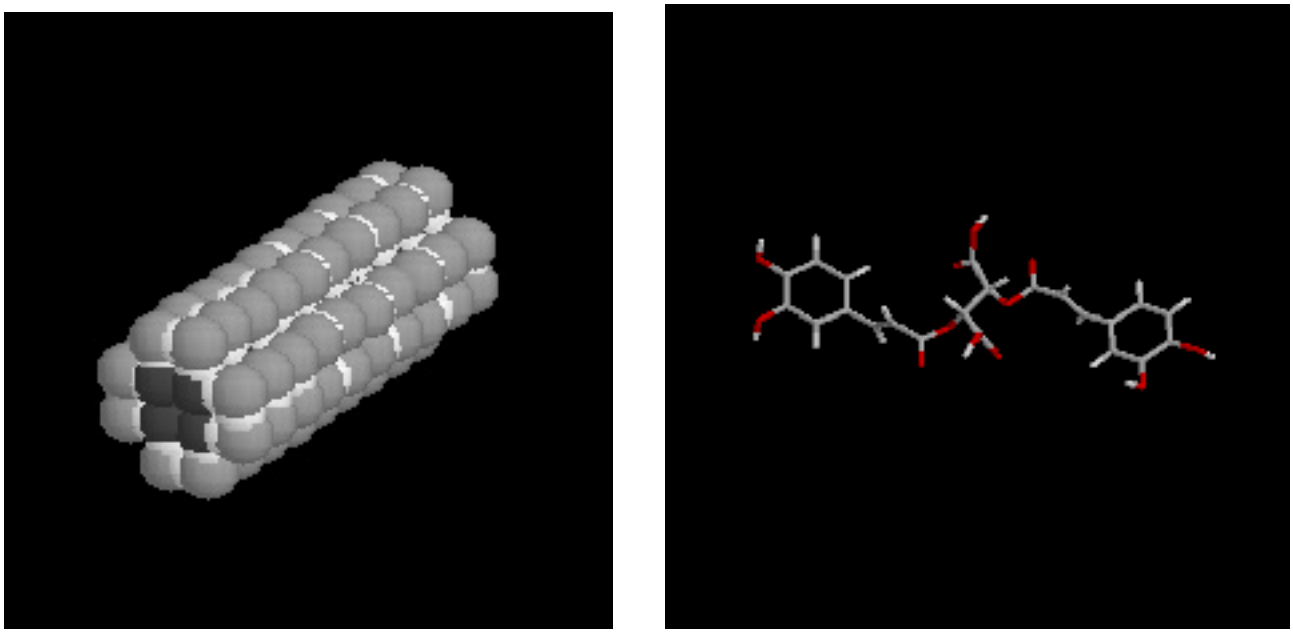

Figure 4: Examples of molecular conformations found by the algorithm. (LEFT) Globally optimal Cα conformation of $\{\text{PHP}\}^{40}$ produced by the program under the HP cubic lattice model [4]. (RIGHT) Molecular conformation of L-chicoric acid produced by the program under the Dreiding atomic force field [13] with implicit solvent [12].

3.2 Comparative Performance

Table 1 shows the algorithm working under the HP cubic lattice model compared to previously published HP cubic lattice methods [1, 5, 15, 16] on 10 previously published HP sequences [16]. The sequences are all 48 HP residues long, resulting in a search space size of $5^{46} \approx 10^{32}$. The sequences, as well as the scores shown for Native, PNS, and HZ, are taken from reference [16]. The score for CGA is taken from the first conformation produced by the multi-queue algorithm; it uses CGA [1] with $m = 6$ to choose the split point when splitting hyper-rectangles, and so its first conformation is identical to CGA (up to tie-breakers). Times for the multi-queue algorithm to produce the best score are shown in parentheses below its score. Run times were not published for Native, PNS, or HZ [5, 15, 16]. The CGA conformation was always produced in 4-5 seconds by the multi-queue algorithm. The algorithm is implemented in C++. Experiments were run on a Pentium 933MHz PC with 256 MB of RAM.

3.3 Growth in Run-Time

Run-time for any known algorithm on any NP-hard problem always scales exponentially if the algorithm is complete and optimal. Consequently, for practical problems the actual numeric value of the exponent is very important. Fig. 5 shows that by separating the time at which the best solution is found from the time at which the proof of optimality is achieved, we are able to reduce the exponent in the scaling relationship. For conformational search in small organic molecules, and for motif search in protein-DNA binding sites, the figure shows the time at which (a) the best conformation was found, (b) the optimality proof was found, or (c) search was abandoned if no proof was obtained.

Table 1: HP model comparison. Each entry is the score found by the method (rows) on the sequence (columns). Higher scores are better. Hours to the best score for the multi-queue branch-and-bound method of this paper ("MQ") are shown in parentheses under its score. "Seq 1-10" are reference 48-residue HP sequences [16]. "Native" is a systematic hydrophobic core assembly process [15], which always finds the global optimum. "PNS" is the inverse folding method used to design the sequences. "HZ" is a method based on hydrophobic zippers [5]. "CGA" is the Bornberg-Bauer chain-growth algorithm [1].

Trial	Seq 1	Seq 2	Seq 3	Seq 4	Seq 5	Seq 6	Seq 7	Seq 8	Seq 9	Seq 10
Native	32	34	34	33	32	32	32	31	34	33
PNS	30	30	31	30	30	30	31	31	30	30
HZ	31	32	31	30	30	29	29	29	31	33
CGA	21	26	23	24	23	20	15	24	23	23
MQ	29	32	32	32	32	29	31	30	32	30
(hours)	(0.24)	(4.43)	(1.70)	(4.80)	(4.27)	(0.48)	(3.36)	(3.58)	(1.53)	(0.28)

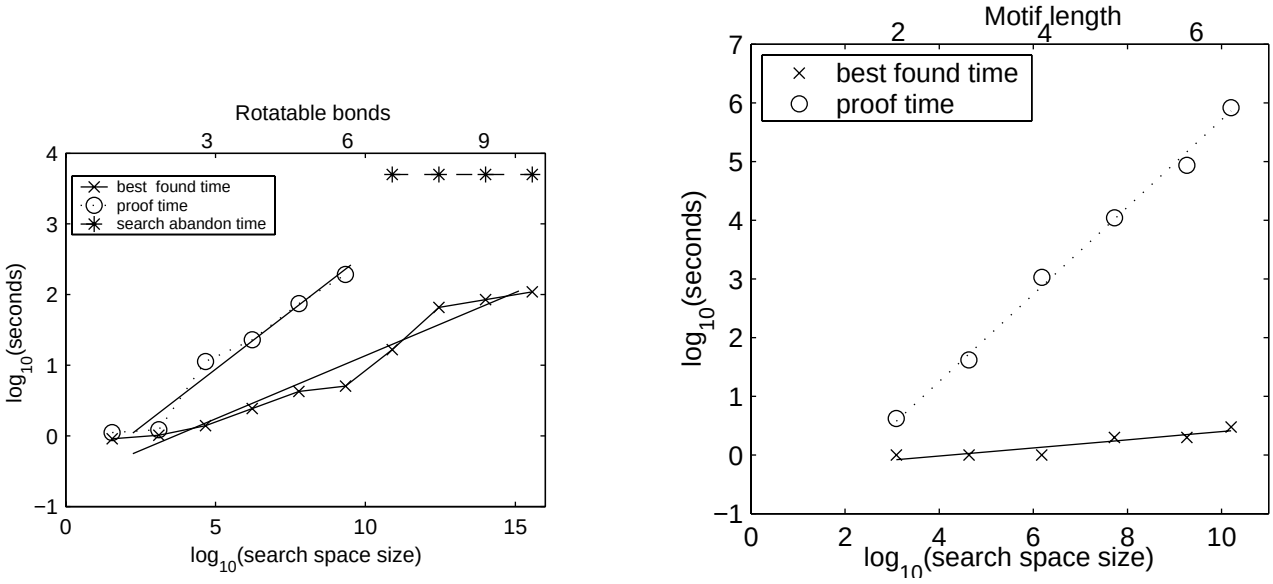

Figure 5: Timing results on problems of increasing size. The x axis shows $\log_{10}$(search space size) and the y axis shows $\log_{10}$(seconds) for the time to find the best result ("X"), the time to proof of global optimality (open circles), or the time at which search was abandoned if no proof was found ("*"). Best-fitting regression lines are shown for best-found and proof times. (LEFT) Small organic molecule conformational search. (RIGHT) Protein-DNA binding motifs for AP-1.

On the left is the time to best conformation and to optimality proof for small organic molecules ranging from one rotatable bonds to ten rotatable bonds. Since the precision for a torsion angle was 10 degrees, the search space size for a molecule with n rotatable bonds is 36^n. The slope of the best-fitting regression line in the figure corresponds to the exponent in the scaling relationship. For the best found time the slope was 0.17, while for the proof time the slope was 0.31.

On the right is the time to the best weight matrix found for a known protein-DNA binding motif. The sequence set consisted of seven DNA sequences known to contain AP-1 binding sites. Each sequence was 120 base pairs long. Within these the program searched for motifs of length two through seven. To limit the resolution and therefore the search space, each weight was limited to values of

$n/4$, where n was an integer value in the range [0,4]. The sum of the weights at each motif position was constrained to sum to 1.0, yielding 35 legal combinations of weights. The search space size was therefore 35^L, where L is the motif length. The slope of the best-fitting regression line for the best found time was 0.069, while the slope for proof time was 0.743.

3.4 Anytime Behavior

Fig. 6 shows the anytime behavior of the algorithm for conformational search in proteins under the HP model and for small organic molecules under an atomic force field. It gives the time and score of the initial conformation found and of the incremental improvements made as the run continued. To make these comparable across different run-times and scores, they are shown as the fraction of the best score found or of the total run-time, respectively; 1.0 represents the best score or the length of the run. The fraction of running time is shown as the log in order to spread out the x axis toward the left, because the program improves its initial score so rapidly that otherwise the graphs all pile up on the left-hand side and details of the trend are difficult to discern.

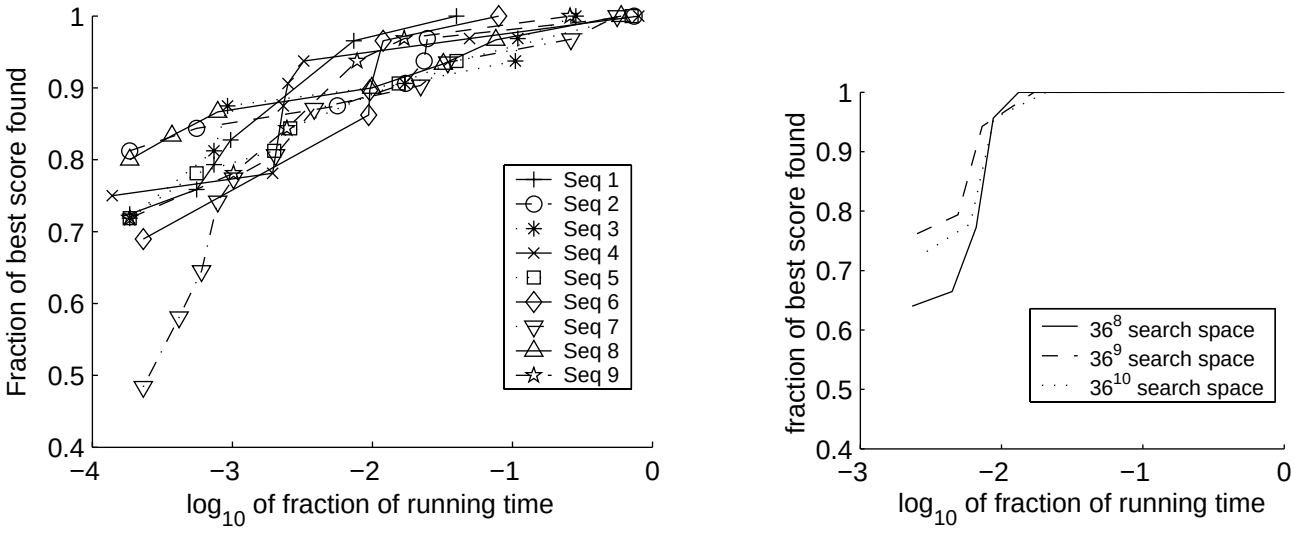

Figure 6: Anytime behavior. The algorithm produces a good result quickly, then incrementally improves that result over time. The time and score of the initial and each improved result is shown. The x axis shows the time as a fraction of elapsed run-time (0.0 corresponds to the start, 1.0 corresponds to the end) on a $\log_{10}$ scale. The y axis shows the score as a fraction of the best score achieved. (LEFT) Each of the 10 sequences in Table 1 (1.0 corresponds to the score shown in the "MQ" row). (RIGHT) The three largest molecules from the left side of Fig. 5; for the smaller molecules the global optimum was always the initial conformation found.

4 Discussion

This paper presented a method for complete, optimal, anytime, and irredundant search in large biological search spaces, and showed it working on several hard problems of biological importance. The program was shown to produce molecular conformations both on- and off-lattice, under both a simple contact potential and an atomic force field with implicit solvation, and to find DNA sequence motifs under a standard log-likelihood metric. Performance compared favorably with previously published methods on published test cases. Producing the global optimum was shown to have a smaller slope and scaling exponent than producing the proof of global optimality. The program exhibited excellent anytime behavior, producing a good result quickly and improving it rapidly with more time.

Although the results presented in this paper are quite preliminary, they are very encouraging. For many problems of biological importance, no provable global optimum has ever before been obtained for realistic problem sizes. Where global optima have been obtained for some problems, invariably this has been accomplished only with great difficulty.

The problems addressed are very difficult and the search space sizes rapidly become formidably large, but we believe that these approaches have the potential to scale to problems of realistic size and importance. Further improvements in the algorithms, the lower bounds, and the choice of split points will be necessary to allow the method to succeed on larger problems. Further research is underway, and we already can see many apparent improvements that will be the focus of future work.

References

[1] Bornberg-Bauer, E., Chain-growth algorithms for HP-type lattice proteins, *Proc. Conf. on Computational Molecular Biology (RECOMB'97)*, ACM Press, New York, 47–55, 1997.

[2] Bryant, S.H. and Lawrence C.E., An empirical energy function for threading protein sequence through the folding motif, *Proteins: Struct., Funct., and Genet.*, 16:92–112, 1993.

[3] Cherkassky, B. and Goldberg, A., An efficient implementation of a scaling minimum cost-flow algorithm, *Journal of Algorithms*, 22:1–29, 1997.

[4] Dill, K. A., Bromberg, S., Yue, K., Fiebig, K. M., Yee, D. P., Thomas, P. D., and Chan, H.S., Principles of protein folding: a perspective from simple exact models, *Protein Science*, 4:561–602, 1995.

[5] Fiebig, K.M. and Dill, K.A., Protein core assembly processes, *J. Chem. Phys.*, 98(4):3475–3487, 1993.

[6] Godzik, A., Kolinski, A., and Skolnick, J., Lattice representations of globular proteins: How good are they?, *J. Computational Chemistry*, 14(10):1194–1202, 1993.

[7] Goldberg, A., The code is available at http://www.star-lab.com/goldberg/soft.html, 1999.

[8] Kanal, L.N. and Kumar, V., *Search in Artificial Intelligence,* Springer-Verlag, Berlin, 1988.

[9] Kittel, C., *Introduction to Solid State Physics, 7th ed.*, John Wiley & Sons, New-York, 1996.

[10] Lathrop, R.H., An anytime algorithm for gapped block protein threading with pair interactions, *Proc. Conf. on Computational Molecular Biology (RECOMB'99)*, ACM Press, New York, 238–249, 1999.

[11] Lathrop, R.H., An anytime local-to-global optimization algorithm for protein threading in $\mathcal{O}(m^2n^2)$ space, *J. Computational Biology,* Fall-Winter, 6(3-4):405–18, 1999.

[12] Lazaridis, T. and Karplus, M., Effective energy function for proteins in solution, *Proteins: Struct., Funct., and Genet.*, 35(2):133–152, 1999.

[13] Mayo, S.L., Olafson, B.D., and Goddard, W.A., A generic force field for molecular simulations, *J. Phys. Chem.*, 94:8897–8909, 1990.

[14] Press, W.H., Teukolsky, S.A., Vettering, W.T., and Flannery, B.P., *Numerical Recipes in C, 2nd ed.*, Section 6.2, 220–221, Cambridge University Press, Cambridge, 1992.

[15] Yue, K. and Dill, K.A., Forces of tertiary structural organization in globular proteins, *Proc. Natl. Acad. Sci. USA,* 92:146–150, 1995.

[16] Yue, K., Fiebig, K.M., Thomas, P.D., Chan, H.S., Shakhnovich, E.I., and Dill, K.A., A test of lattice protein folding algorithms, *Proc. Natl. Acad. Sci. USA*, 92:325–329, 1995.