MRfY: Remote Homology Detection for Beta-Structural Proteins Using Markov Random Fields and Stochastic Search

Noah M. Daniels

Andrew Gallant

Norman Ramsey

Lenore J. Cowen

*
Dept. of Computer Science
Tufts University
161 College Ave
Medford, MA 02155

ABSTRACT

We introduce MRfY, a tool for protein remote homology detection that captures beta-strand dependencies in the Markov random field. Over a set of 11 SCOP beta-structural superfamilies, MRfY shows a 14% improvement in mean Area Under the Curve for the motif recognition problem as compared to HMMER, 25% improvement as compared to RAPTOR, 14% improvement as compared to HHPred, and a 18% improvement as compared to CNFPred and RaptorX. MRfY was implemented in the Haskell functional programming language, and parallelizes well on multi-core systems. MRfY is available, as source code as well as an executable, from <http://mrffy.cs.tufts.edu/>.

Categories and Subject Descriptors

J.3 [Life and Medical Sciences]: Biology and genetics

General Terms

Algorithms

1. INTRODUCTION

Computational approaches to the problem of recognizing remote homologs of well-annotated protein structures from sequence have emerged as some of the most successful high-throughput strategies for developing hypotheses about the functions of unknown proteins. Recognition of remote homologs from protein sequence is a challenging problem, however, particularly for β -structural motifs, where many of the residues whose interactions drive the fold can be a variable distance, and sometimes a long distance apart in sequence [20]. Many of the popular sequence-based methods for recognizing close homologs, such as Profile Hidden Markov Models

(HMMs) [7, 11], perform particularly poorly on β -structural motifs, precisely because the HMM model is not powerful enough to capture these long-range dependencies [17, 33, 23, 3, 28]. Several ways to generalize these HMMs to Markov random fields (MRFs) have therefore been proposed [31, 16, 29, 8, 20, 24], but with the additional power of the random field comes several computational challenges. In particular, there is a trade-off: as the complexity of the random field increases, so does its predictive power, but so too do the required amount of training data and the computational complexity of aligning a query sequence to the model.

In 2010, Menke, Berger and Cowen introduced SMURF [20], a Markov random field method designed to recognize protein sequences that fold into β -propeller shapes. While SMURF greatly improved β -propeller recognition over existing HMM and threading methods, it was actually quite simple as far as MRFs are concerned: it combined a score based on a standard profile hidden Markov model with a conditional probability score that incorporated the statistical preferences of the amino acid residues that are hydrogen-bonded in β -sheets, thereby capturing some of the strongest long-range dependencies in β -structural motifs. The pairwise statistical preferences were learned from solved β -structures across the entire PDB, so there was sufficient training data. However, there was a great computational cost: the minimum-energy alignment of the sequence to the Markov random field model was generated exactly using a doubly-exponential multi-dimensional dynamic program.

This computational cost prevented the SMURF MRF from being applied beyond propellers to other β -structures. When run on β -barrels, or other β -structures with even moderately complex strand interleaving patterns, the exact computation of the SMURF MRF score becomes intractable. There are two ways to mitigate the computational bottleneck. One approach is to simplify the random field, and only consider some of the pairwise statistical preferences for the β -strands, namely those that are more local in sequence. This is the methodology we introduced in a 2012 paper of Daniels, Hosur, Berger and Cowen, when we designed SMURFLite [6]. The other approach is to retain the entire SMURF MRF, but consider stochastic search approaches to heuristically explore alignments of the sequence to the MRF, in order to find low-energy but not necessarily minimum-energy parses. This is the approach we take in the present work.

In particular, we introduce MRfY, a suite of methods to perform stochastic searches of alignments to the SMURF MRF. We con-

*Corresponding author: lenore.cowen@tufts.edu

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

BCB '13, September 22 - 25, 2013, Washington, DC, USA
Copyright 2013 ACM 978-1-4503-2434-2/13/09 ...\$15.00.

Finally, we find the best approach is a hybrid combination of SMURFLite and MRFy: we initialize the MRFy stochastic search from the best sequence parse that SMURFLite can find, and then run the stochastic search from there. On the same set of β -barrel superfamilies considered in [6], we show an improvement of 3.4% in the mean (3.6% in the median) Area Under Curve (AUC) for β -structural motif recognition as compared to the SMURFLite results in [6]. For the same benchmark set, MRFy initialized using SMURFLite shows an improvement of 14% in the mean (15% in the median) over HMMER ([7]), an improvement of 25% mean (40% median) as compared to RAPTOR ([32]), an improvement of 18% mean (21% median) over CNFPred [12] and RaptorX [25], and an improvement of 14% in the mean (23% median) in AUC over HHPred ([26]). For clarity, we note that these summary statistics come from first computing the mean and median AUC for each method separately, and then reporting the overall improvement in both statistics achieved by MRFy; a full superfamily-by-superfamily comparison appears in Table 4.

given the transformations:

$$\begin{aligned} a'_{ss} &= -\log a_{ss} \\ e'_s(x) &= -\log \frac{e_s(x)}{q_x} \\ V_j^M(i) &= -\log V_j^M(i) \\ P'(x_i|x_{\pi j}) &= -\log P(x_i|x_{\pi j}) \end{aligned}$$

This is exactly the recurrence relation that SMURF [20] and SMUR-FLite [6] solve using multidimensional dynamic programming. Note that gaps are prohibited within β -strands, ensuring that if x_i and $x_{\pi j}$ are hydrogen-bonded, and x_{i+2} and $x_{\pi j+2}$ are hydrogen-bonded in the same β strands, then $x_{\pi j}$ and $x_{\pi j+2}$ are exactly two amino acids apart in sequence. The *interleave* of a pair of matched β -strands is defined as the number of other β -strands that occur between those two strands in the protein sequence, plus one. As demonstrated by Daniels, et al. [6], as the maximum interleave of a pair of matched β -strands increases, the computational complexity of computing the best score for the associated SMURF MRF grows exponentially.

As an alternative to exactly solving these more complex recurrence relations, we might consider a divide-and-conquer approach. Each β -strand can be thought of as breaking the larger model into two smaller models; collectively, all the β -strands divide the Markov random field into many small, *independent* hidden Markov models. Thus, for any particular path through the Markov random field, corresponding to a particular placement of query sequence residues onto the nodes of the model, we could compute the augmented Viterbi score by summing the Viterbi scores of each smaller hidden Markov model, along with the contribution to the Viterbi score from the β -strands.

Since only match states are allowed for β -strand residues, the contribution of each such residue is only:

$$\begin{aligned} V_j^M(i) &= e'_{M_j}(x_i) + \\ &V_{j-1}^M(i-1) + a'_{M_{j-1}M_j} + P(x_i|x_{\pi j}) \end{aligned}$$

The asymptotic complexity of the Viterbi algorithm is $O(mn)$, where m is the length of the model and n is the length of the query sequence. Furthermore, the asymptotic complexity of the beta-strand contribution to the Viterbi score for a particular placement of residues is just $O(b)$, where b is the combined length of the β -strands.

Thus, a new algorithm for computing the optimal path through a Markov random field for a given query sequence presents itself. Since we require that every β -strand position be occupied by a residue (as we force those positions into match states), we could simply consider every possible assignment of a residue to a β -strand, computing the score for each one, and choose the best-scoring placement.

Metaphorically, we can picture the residues of the query sequence as beads, and the Markov random field as the string of a necklace. The β -strands can be thought of as particular substrings of the string that must be covered by beads, while non- β regions may be exposed (resulting in delete states in the model). To continue the metaphor, we may jam extra beads onto non- β regions of the string, resulting in insert states in the model. Given that the beads already have a specified order, we must consider all the ways to slide the beads up and down the string such that all of the β -regions are covered. Since the regions between β -strands can have their contribution to the score computed according to the Viterbi recurrence relations, we need only consider all the unique ways to assign residues to the β -strand nodes.

However, it is easy to show that there are an exponential number of such assignments.

Let a Markov random field model (N, B) be defined as a sequence N of nodes $n_i, i \in (1..m)$, and a sequence B of β -strands $b_i, i \in (1..k)$. Each β -strand has length l_i , and contains a subsequence of the nodes N . This subsequence is determined by the specifics of the model, which can be referred to as $b_{ij}, i \in (1..m), j \in (1..l_i)$. Let a query sequence be defined as a sequence R of residues $r_i, i \in (1..n)$. Let $L = \sum_{i, i \leq k} l_i$.

There are:

$$\prod_{i \in (1..k)} n - L - (i \times l_{max}) = (n - 2L - k \times (l_{max}))^{\lceil \frac{k}{2} \rceil}$$

possible placements of R onto (N, B) . Asymptotically, as n grows, this is dominated by n^k , leading to an asymptotic complexity of $O(n^k)$.

A typical Markov random field might have 10 or 20 β -strands, and a typical protein query sequence might have between 300 and 600 residues. Thus, if we wish to consider all possible paths through a Markov random field for a protein sequence, we must consider as many as $600^{10} \approx 6 \times 10^{27}$ possible paths through the model.

2.2 Stochastic search

Since an exhaustive search for an optimal alignment of a protein sequence to a Markov random field is exponential in complexity, we turn to stochastic search to mitigate this complexity.

Framed as an optimization problem, MRFy, like SMURF, seeks to minimize the augmented Viterbi score, which equates to maximizing probability (recall that this score is the negative log of a probability). SMURF [20] finds this minimum exactly, using multidimensional dynamic programming, which is exponential in the interleave number of beta strands. MRFy, in contrast, uses stochastic search, as described next.

Given a placement of query-sequence residues into β -strand nodes of the Markov random field, the score can be computed exactly. Thus, the search space is the set of all possible ways to place residues on these nodes. MRFy, as a package, implements many different stochastic search strategies for finding the optimal alignment of a query sequence to the Markov random field. A particular MRFy strategy involves specifying three different criteria: first, how the initial guess is generated (four options), second, how the stochastic search proceeds (we include a simulated annealing, genetic algorithm, and local search option), and finally, when to terminate and output the search and output the best alignment seen thus far (three options). We first describe the initial guess options, then the termination options, and finally the options for stochastic search implemented in MRFy.

We explored four models for generating initial guesses for our search techniques. A *random-placement* model uniformly positions the β -strands along the query sequence, under the constraint that only legal placements may be generated, and thus the placement of any β -strand must leave room for all the other β -strands in the model.

A *secondary-structure* prediction model uses the PSIPRED [19] secondary-structure prediction program to determine the positions of β -strands. Given a PSIPRED prediction for the secondary structure of a query sequence, we place β -strands at the most likely locations according to this prediction profile, randomized by a small amount of noise.

A *template-based model* is based on the observation that true homologs to a structurally-derived template should have their β -strands in very roughly similar places, in sequence, to the proteins that made up that template. This will not always hold, but appears to provide for reasonable initial guesses. Given the position

of each β -strand within a template Markov random field, we scale the query sequence linearly (as it may be shorter or longer than the model) and place the β -strands in scaled positions. Note that we do not scale the β -strands themselves; their lengths are preserved. We scale only the distances between β -strands. We inject a small amount of noise into the placements, so that population-based models, such as multi-start simulated annealing and genetic algorithms, start with heterogeneous solutions. **This is the initial guess method that we used to generate the results in Tables 1, 2, and 3 in Figure 5, and the MRFy¹ and MRFy² results in Table 4.**

Finally, we implemented a model that attempts to unify the SMURFLite and MRFy approaches. Given a template T , we create a new template, T' , exactly as in [6], from which we have removed any β -strand pairs that exceed an interleave threshold of 2 (that is, any pairs that have more than two other β -strands between them in sequence). We then run SMURFLite on T' , to create an alignment a' . Note that a' is an optimal solution to the template T' . We wish to use a' as an initial guess for MRFy, but we must first add in the β -strands that were removed. In order to do so, we greedily move the placement of each β -strand in a' as needed to make room for those other β -strands. **The resulting alignment a becomes the initial guess for MRFy. This is the initial guess method that we used to generate the results for MRFy³ in Table 4.**

Since we do not know how to determine when a stochastic search process has found a *global* optimum (as opposed to a good local optimum), we must also have some termination criterion for the search. We implemented three alternative termination criteria. The first is a simple generation-counting approach, where the search terminates after a user-specified number of generations. The second is a time-based approach, where the search terminates after a user-specified amount of time has elapsed. Finally, a *convergence* model terminates after the search has failed to improve after a user-specified number of generations.

In practice, these criteria are easily combined, with a convergence approach often halting searches early with good results, while the generation- or time-based limit ensures that the search does not take longer than a user is willing to wait. **We test different termination conditions in Tables 1, 2, and 3; the results in Table 4 and Figure 5 are all generated with a fixed 30 second time limit.**

We next describe the alternative heuristics that MRFy implements for stochastic search: simulated annealing, a genetic algorithm, and a local search strategy.

2.2.1 Simulated Annealing

Simulated annealing [13] is a heuristic for stochastic search, inspired by the physical process of annealing in metals. Whereas a simple hill-climbing approach will always terminate at the first local optimum encountered, simulated annealing introduces an *acceptance probability function*:

$$P(e, e', T) = \begin{cases} 1, & \text{if } e' < e \\ \exp(-(e' - e)/T), & \text{otherwise} \end{cases}$$

where $e = E(s)$
 $e' = E(s')$

which relies on some energy function $E(s)$ of the current state s and a candidate state s' , and a temperature function T that tends towards zero as the search progresses. In our implementation, we used an exponentially-decaying temperature function:

$$T(t) = k^t \times T_0$$

given time t , initial temperature T_0 , and a constant k . The motivation for this decaying temperature function is that, as time pro-

gresses, the likelihood of being in a *poor* local optimum lessens, and thus, the closer to random hill-climbing we would like the search to behave.

Our energy function $E(s)$ is, naturally, the augmented Viterbi score of a placement:

$$E(s) = V_m^M(n)$$

where m is the final residue in the query sequence and n is the final node in the Markov random field, and the β -strand placements are determined by s .

We implemented simulated annealing in MRFy according to this model. We also implemented a *multi-start* version of simulated annealing in MRFy, where a set of independently-generated guesses is subject to simulated-annealing random descent, in parallel. At the termination of the search, the best solution from among all the candidates is chosen.

2.2.2 Genetic Algorithm

A genetic algorithm [10] relies on the idea of *selection* among a population of varied solutions to an optimization problem. At each of many generations, the fitter individuals in the population—those solutions which exhibit more optimal scores—are allowed to continue into the next generation. Not only do they continue into the next generation, but they are allowed to “reproduce,” or recombine, to produce new solutions. A particular solution to a problem, within the context of a genetic algorithm, is called a *chromosome*. At each generation, some fraction of the fittest solutions are selected and randomly paired with one another. Each pair of solutions produces one or more offspring; each offspring is the result of two steps: *crossover* of the two chromosomes, followed by random *mutation* of the offspring. The mutation is nondeterministic; the crossover may be deterministic or nondeterministic. The resulting offspring, along with their parents, are then evaluated according to the objective function, and this process iterates until some termination condition.

MRFy’s genetic algorithm implementation uses the same representation for a placement as simulated annealing: an ordered list of integers.

Let a *placement* p on a model with k β -strands be an ordered set of integers $p_i, i \in (1..k)$. Given two placements, p and q , MRFy implements crossover of two chromosomes using the following algorithm:

1. Set the new placement, p' , to the empty set.
2. Repeat until all placements have been chosen:
 - (a) Set the position for p'_0 to p_0
 - (b) Set the position for p'_k to q_k
 - (c) Remove p_0, p_k, q_0 , and q_k
 - (d) Adjust indices so p and q now range from what had been 1 to $k - 1$.

Our actual implementation is purely functional, and simply consumes elements from lists. In effect, though, this algorithm simply chooses the ‘left-most’ elements from one parent and the ‘right-most’ elements from another. After crossover, the mutation step simply moves each element p_i of the placement p by a small, random amount, within the constraints imposed by the neighboring β -strands. The motivation behind this approach is to take two solutions that are of high fitness (recall that the worst solutions at every generation are not allowed to contribute to the next generation), and produce a new solution that combines one “half” (roughly) of one

Figure 2: The crossover and mutation process in MRFy's genetic algorithm implementation. Given parent p (black) and parent q (gray), alternate left and right placements from p and q . Then, apply small random mutations to the resulting placement p' .

solution with one “half” of the other. See Figure 2 for an illustration of this procedure.

Given these operations for crossover and mutation, MRFy's genetic algorithm implementation initializes a population of a user-specified size P (typically one thousand placements, though we experimented with as many as ten thousand). In parallel, each placement is scored according to the objective function. Since scoring is far more computationally expensive than crossover and mutation, we allow them all to reproduce, paired at random. We then score them, and choose the P best-scoring placements for the next generation. This process repeats until a termination condition is met, at which point the single best placement is returned. We note that a future enhancement to MRFy could return the k best placements for some user-specified threshold k , if multiple high-scoring alignments were to be considered.

2.2.3 Local Search

Constraint-based local search [9] is a family of approaches for exploring “neighborhoods” in feature space in a randomized manner, subject to the constraints of that solution space. In the context of MRFy, the constraints are the previously-discussed restrictions that β -strands cannot overlap, and every residue must be placed in a β -strand. Given a single candidate solution, local search explores the immediate neighborhood in great detail (perhaps, but not necessarily exhaustively). When the local search cannot escape a local optimum, then some sort of *non-local* move may be attempted.

This non-local move may rely on a population-based diversification approach, in which parts of the solution may change dramatically. In a sense, local search bears some resemblance to a genetic algorithm, except that a population of solutions is created only when the search is stuck in a local optima, and the best solution in that population is chosen for a new search.

In MRFy's implementation, each step in the search consists of two phases: *diversification* (See Figure 3) and *intensification*. The diversification algorithm is as follows: Begin with a candidate solution s (a placement), which is just an ordered list of integers. Given s , break the list into three sub-lists s_0, s_1, s_2 , at randomly-chosen boundaries. Choose one of the sub-lists s_i at random, and mutate it into k copies s_{i1} through s_{ik} at random, for some user-defined value of k (we used $k = 10$), within the constraints imposed by the other sub-lists and the lengths of the β -strands. Re-combine each set of lists, (s_{1j}, s_{2j}, s_{3j}) into a new placement s'_j , $j \in \{1..k\}$. Score each placement s'_j , return the best-scoring of the k new placements as a new solution.

Once diversification produces a new candidate solution, inten-

sification brings it toward a local maximum. The intensification algorithm is as follows: Begin with a candidate solution s . Until no better-scoring placements are generated, repeat the following steps: For each element $e \in s$, generate four new placements s'_{i1} through s'_{i4} by moving e up and down by 1 and 2, as long as those moves do not violate the constraints. Next, score each candidate placement s'_{ij} . Finally, set s to the best-scoring candidate placement s'_{ij} . Upon termination, return s as a new solution. **Based on the comparative results of different stochastic search methods in Tables 1, 2, and 3, we used local search to generate all MRFy results in Table 4 and Figure 5.**

Figure 3: The diversification step in local search. In this example, β -strand placements in the middle of the query sequence, corresponding to residue positions 24, 31, and 47, are varied at random within the bounds of the surrounding placements (corresponding to residue positions 17 and 56) in order to explore a diverse areas of the search space from a locally optimum solution.

2.3 Evaluating search strategies

As MRFy supports three significantly different stochastic search strategies, and a number of tunable parameters such as termination conditions and (for simulated annealing) the cooling schedule, we conducted a search over parameter space using a small data set. We built Markov random field templates from the fold “8-bladed Beta-Propellers”, and the superfamilies “Barwin-like endoglucanases” (a β -barrel superfamily) and “Concanavalin A-like lectins/glucanases” (a β -sandwich superfamily). We were interested in the speed of convergence for a true-positive test case, so we tested each template with a protein sequence chosen from that fold or superfamily: for the 8-bladed propeller, we chose ASTRAL chain d1lrwa_ (Methanol dehydrogenase, heavy chain from *Paracoccus denitrificans*). For the barwin-like endoglucanases, we chose ASTRAL chain d2pica1 (Membrane-bound lytic murein transglycosylase A, MLTA from *E. coli*). For the lectins/glucanases, we chose ASTRAL chain d2sbaa_ (Legume lectin from soy bean (*Glycine max*)).

We tested simulated annealing with a population size of 10, a maximum number of generations of 10000, convergence periods of 200, 500, and 1000 generations, and a cooling factor of 0.99 (preliminary tests showed little impact from varying the cooling factor among 0.9, 0.99, and 0.999).

We tested the genetic algorithm implementation with a population size of 1000 and 10000, a maximum number of generations of 500, and convergence periods of 10, 50, and 100.

Since the local search distinguishes between diversification and

intensification, counting the number of generations is ambiguous; we used a time limit of 10 seconds, 30 seconds and 5 minutes. All tests were conducted on a 12-core AMD Opteron 2427 with 32GB RAM, devoting all 12 cores to MRFy. For each test, we report statistics based on ten runs for each set of parameters.

2.4 Simulated Evolution

In MRFy, we incorporated precisely the same “simulated evolution” implementation, as first proposed by Kumar and Cowen [14, 15], as we did for SMURFLite in [6]. We added pairwise mutations based on β -strand pairings. We use the same mutation frequencies as in [6]. For each training sequence, we generate 150 new artificial training sequences. For each of these artificial sequences, we mutate at a 50% mutation rate per length of the β -strands. The goal of this approach is to partially compensate for limited training data, particularly in superfamilies that contain relatively few distinct sequences.

2.5 Datasets

From SCOP ([21]) version 1.75, we chose the same β -barrel superfamilies as [6]. These superfamilies were: “Nucleic acid-binding proteins” (50249), “Translation proteins” (50447), “Barwin-like endoglucanases” (50685), “Cyclophilin-like” (50891), “Sm-like ribonucleoproteins” (50182), “PDZ domain-like” (50156), “Prokaryotic SH3-related domain” (82057), “Tudor/PWWP/MBT” (63748), “Electron Transport accessory proteins” (50090), “Translation proteins SH3-like domain” (50104), and “FMN-binding split barrel” (50475).

2.6 Training and testing process

For the β -barrel superfamilies, we performed strict leave-family-out cross-validation. We built training templates at the superfamily level. For each superfamily, its constituent families were identified. Each family was left out, and a training set was established from the protein chains in the remaining families, with duplicate sequences removed. We built an MRF on the training set, both with and without training-data augmentation using the same “simulated evolution” implementation as [6]. We chose protein chains from the left-out family as positive test examples. Negative test examples were protein chains from all other superfamilies in SCOP classes 1, 2, 3 and 4 (including other barrel superfamilies), indicated as representatives from the nr-PDB ([2]) database with non-redundancy set to a BLAST E-value of 10^{-7} .

We used MRFy’s local search mode (see Section 2.2.3) to align each test example to the trained MRF. The score reported for MRFy was the combined HMM and pairwise score from the MRF, which is identical to the SMURF energy function. For each training set, the scores for both methods (MRFy with and without simulated evolution) were collected and a ROC curve (a plot of true positive rate versus false positive rate) computed. We report the area under the curve (AUC statistic) from this ROC curve [27].

3. RESULTS

3.1 Search strategies

For the three stochastic search approaches, we compared the raw score achieved by each approach under a variety of conditions, as discussed in Section 2.3. The raw score is simply the negative log of the probability of the best path found through the model. Thus, raw scores are not comparable between models, but they are comparable between query sequences for a given model.

We explored the different MRFy stochastic search options in depth on three structures: the 8-bladed propellers, the Barwin-like

endoglucanase β -barrel superfamily, and the Concanavalin A-like lectins/glucanases. We chose these three structures in part because they are structures on which computing the global optimum score, while slow, is actually tractable. For the 8-bladed propeller fold, SMURF itself returns the best-scoring alignment. The Barwin-like endoglucanase β -barrel superfamily has an unusually low interleave number for a β -barrel superfamily, namely an interleave number of 4. While SMURFLite is never recommended to be run with an interleave number greater than 2, it is sometimes tractable (though very slow) when run with an interleave threshold of 4, and when it is run with an interleave threshold with 4 on a structure with a maximum interleave number of 4, it returns the global optimum (though for this structure this requires 10 minutes per alignment on a high-end server). The Concanavalin A-like lectins/glucanases superfamily has too great a maximum interleave number for SMURFLite to be tractable when set to run on the entire MRF, but because of the relatively small number of β -strands, we were able to exhaustively enumerate every possible placement of β -strands, and thus check every possible alignment to achieve the globally optimal score.

Table 1 indicates the performance of different stochastic search techniques on the 8-bladed β -propeller fold, using the *template-based* model for initial guesses described in Section 2.2. While the simulated annealing and genetic algorithm approaches exhibit less variance (a smaller standard deviation) from run to run, they do not approach the minimum score of the local search approaches. Multistart simulated annealing with a population of 10 and a convergence threshold of 200 generations averages 29.3 seconds per search, but only achieves a minimum score of 2112, in contrast to a global optimum of 1781, a ratio of 0.843, though it converged in all cases.

In contrast, local search, given 30 seconds, achieves a minimum score of 1982 (a ratio of 0.899), and even in only 10 seconds achieves a minimum score of 1992 (a ratio of 0.894). However, the global minimum score of 1781, which is achieved by SMURF on the 8-bladed β -propeller template, is only reached by MRFy with local search two out of ten times, and this result required local search be allowed to run for twenty minutes. Thus, for this problem domain, local search seems to outperform our simulated annealing and genetic algorithm implementations.

Table 2 indicates the performance of the stochastic search techniques on the “Barwin-like endoglucanases” β -barrel superfamily, again using the template-based model for initial guesses. These structures are less complex than the propellers, even though they are *more* computationally complex for SMURFLite [6] if an interleave threshold greater than 2 is used. We see less variance than with the propellers, but once again, the local search technique achieves a lower minimum score than simulated annealing or the genetic algorithm.

Notably, local search achieves a minimum score of 978, which a SMURF alignment indicates to be a global minimum for this sequence on this template. With a time limit of 10 seconds, local search found this global minimum in one out of ten runs. With a time limit of 30 seconds, local search found it in two out of ten runs, and with a time limit of 5 minutes, in four out of ten runs.

Table 3 indicates the performance of the stochastic search techniques on the “Concanavalin A-like lectins/glucanases” β -sandwich superfamily. These structures are also more complex than the propellers, even though they are also more computationally complex for SMURFLite with an interleave threshold greater than 2. On this superfamily, there is a closer overlap between the minimum score achieved by simulated annealing, at 790 (a ratio of 0.701 compared to the global minimum), and the range seen by local search; local search with a time limit of 30 seconds achieves a *mean* minimum

Table 1: Stochastic search performance on 8-bladed β -propeller

	Min Score	Mean Score	Std Score	Mean Time (s)
SA 200	2112	2139	12.2	29.3
SA 500	2129	2146	9.3	1020
SA 1000	2112	2130	7.8	3314
GA 1000/10	2105	2126	6.6	285
GA 1000/50	2094	2118	7.7	1239
GA 1000/100	2107	2120	3.8	548
GA 10000/10	2087	2111	7.2	5809
GA 10000/50	2094	2112	7.1	5174
GA 10000/100	2079	2114	9.0	10226
LS 10s	1992	2015	19.4	10
LS 30s	1982	1991	10.9	30
LS 5m	1818	1876	37.2	300

Performance of stochastic search techniques on an 8-bladed β -propeller template. SA is Simulated Annealing, GA is Genetic Algorithm, and LS is Local Search. For Simulated Annealing, we show results for convergence thresholds of 200, 500, and 1000 generations. For the Genetic Algorithm, we show results for convergence thresholds of 10, 50, and 100 generations, and for population sizes of 1000 and 10000.

For Local Search, we show results for time limits of 10 seconds, 30 seconds and five minutes, on a 12-core AMD Opteron. MRFy never achieved the global optimum score of 1781, achieved by SMURF, on this template, except when local search was given 20 minutes of compute time, in which case it found the global optimum two out of ten times. These results were obtained using MRFy without simulated evolution, and with initial guesses provided by the template-based model as described in Section 2.2. These results show that local search outperforms the other search strategies, and presents the tradeoff between the running time and accuracy of stochastic search.

score of 791 (a ratio of 0.700), though its best was 740 (a ratio of 0.748).

Notably, when given a time limit of 5 minutes, local search achieved the *global minimum* of 554 (as determined by exhaustive search) ten out of ten times. Local search never found this score when given only 10 seconds or 30 seconds as a time limit.

Our Haskell implementation made it exceedingly easy to parallelize MRFy across multiple processing cores. By default, MRFy will take advantage of all processing cores on a system; we tested the parallel speedup on a system with 48 processing cores. We measured the run-time performance of MRFy’s genetic algorithm implementation (with a fixed random seed) on the “8-bladed β -propeller” template. The model has 343 nodes, of which 178 appear in 40 β -strands. The segments between β -strands typically have at most 10 nodes. We used a query sequence of 592 amino acids, but each placement breaks the sequence into 41 pieces, each of which typically has at most 20 amino acids. Because MRFy can solve the models between the β -strands independently, this benchmark has a lot of parallelism.

Figure 4 shows speedups when using from 1 to 48 of the cores on a 48-core, 2.3GHz AMD Opteron 6176 system. Errors are estimated from 5 runs. After about 12 cores, where MRFy runs 6 times as fast as sequential code, speedup rolls off. We note that by running 4 instances of MRFy in parallel on different searches, we would expect to be able to use all 48 cores with about 50% efficiency. All three stochastic search approaches demonstrate comparable parallel efficiency; parallelism is achieved both at the population level (in which individual candidate solutions are scored in parallel) and at the solution level (in which the Viterbi score for each independent hidden Markov model, separated by β -strand placements, is scored in parallel). Thus, we would anticipate somewhat better parallel efficiency with a larger number of β -strands, and worse parallel efficiency with few β -strands. It is also worth noting that these experiments were confined to a departmental computing server; while we applied best practices to ensure the significance of these results, they should not be taken to represent an optimally-configured computer system. Better parallel efficiency might be obtained with a different choice of operating system kernel and

memory configuration.

Based on our results described above, we decided to have MRFy default to local search, with a time limit of 30 seconds. We further explore MRFy’s performance with and without simulated evolution, and with initial guesses derived from SMURFLite.

Figure 4: MRFy’s parallel speedup on an 8-bladed β -propeller, using a 48-core system. After about 12 cores, speedup falls off.

3.2 Remote homology detection accuracy

We compared MRFy using local search and a 30-second time limit, against HMMER [7], Raptor [32], HHpred [26], RaptorX [25], and CNFPred [12]. We performed cross-validation testing on 11 β -barrel superfamilies, both with and without simulated evolution. For MRFy, the balance between accuracy and computational efficiency is determined by the termination conditions, as well as the search technique chosen. Because local search so dramatically outperformed simulated annealing and the genetic algorithm at obtaining better-scoring alignments, we conducted these cross-validation tests only on local search, using a simple time limit as a termination condition. We chose 30 seconds as a balance between speed and accuracy; a 5 minute time limit might result in better accuracy, but for high-throughput, whole-genome scans, 5 minutes per alignment is excessive.

Table 2: Stochastic search performance on “Barwin-like” β -barrel

	Min Score	Mean Score	Std Score	Mean Time (s)	Optimal
SA 200	1064	1071	3.8	79.5	0
SA 500	1047	1063	7.6	104	0
SA 1000	1024	1047	14.0	523	0
GA 1000/10	1061	1069	3.6	232	0
GA 1000/50	1059	1066	3.1	442	0
GA 1000/100	1058	1069	4.0	1382	0
GA 10000/10	1058	1063	2.5	8205	0
GA 10000/50	1059	1061	2.2	10306	0
GA 10000/100	1057	1061	2.2	16395	0
LS 10s	978	995	16.2	10	0.1
LS 30s	978	987	6.9	30	0.2
LS 5m	978	981	2.9	300	0.4

Performance of stochastic search techniques on the “Barwin-like endoglucanases” β -barrel template. SA is Simulated Annealing, GA is Genetic Algorithm, and LS is Local Search. For Simulated Annealing, we show results for convergence thresholds of 200, 500, and 1000 generations. For the Genetic Algorithm, we show results for convergence thresholds of 10, 50, and 100 generations, and for population sizes of 1000 and 10000. For Local Search, we show results for time limits of 10 seconds, 30 seconds and five minutes, on a 12-core AMD Opteron. The “Optimal” column indicates the fraction of runs for each search method that achieved the global optimum. These results were obtained using MRFy without simulated evolution, and with initial guesses provided by the template-based model as described in Section 2.2. These results show that local search outperforms the other search strategies, and presents the tradeoff between the running time and accuracy of stochastic search.

Table 3: Stochastic search performance on β -sandwich

	Min Score	Mean Score	Std Score	Mean Time (s)	Optimal
SA 200	795	834	18.6	84.7	0
SA 500	790	820	17.3	192	0
SA 1000	791	811	14.7	493	0
GA 1000/10	874	888	4.1	1869	0
GA 1000/50	878	883	2.5	1305	0
GA 1000/100	865	878	5.6	4309	0
GA 10000/10	872	877	2.5	6999	0
GA 10000/50	875	879	3.1	5317	0
GA 10000/100	869	875	4.5	10733	0
LS 10s	771	826	31.7	10	0
LS 30s	740	791	47.0	30	0
LS 5m	554	554	0.0	300	1.0

Performance of stochastic search techniques on a “Concanavalin A-like lectins/glucanases”, a 12-stranded β -sandwich template. SA is Simulated Annealing, GA is Genetic Algorithm, and LS is Local Search. For Simulated Annealing, we show results for convergence thresholds of 200, 500, and 1000 generations. For the Genetic Algorithm, we show results for convergence thresholds of 10, 50, and 100 generations, and for population sizes of 1000 and 10000. For Local Search, we show results for time limits of 10 seconds, 30 seconds and five minutes, on a 12-core AMD Opteron. The “Optimal” column indicates the fraction of runs for each search method that achieved the global optimum. These results were obtained using MRFy without simulated evolution, and with initial guesses provided by the template-based model as described in Section 2.2. These results show that local search outperforms the other search strategies, and presents the tradeoff between the running time and accuracy of stochastic search.

We compared MRFy’s performance, both with and without simulated evolution, to the results from [6], for exactly the same set of 11 SCOP β -barrel superfamilies that were considered in [6]. Table 4 shows the area (AUC) under the Receiver Operator Characteristic (ROC) curve for MRFy, the very best result from SMURFLite, and HMMER, RAPTOR, HHPred, RaptorX (specifically, Boost-Threader), and CNFPred. Importantly, we are choosing the best SMURFLite parameters for each superfamily, which could not be known in advance; thus, we demonstrate improvements over the *very best* SMURFLite can perform, rather than just an average case.

We first note the “Barwin-like endoglucanases” superfamily highlighted in [6]. SMURFLite performed better as the interleave threshold was increased on this superfamily, and also when simu-

lated evolution was added. Since MRFy discards no β -strands, we were curious how it would perform on this superfamily. Notably, this superfamily has exceedingly little training data; during cross-validation, there are at most 4 training sequences and as few as 3 when filtered at a BLAST E-value of 10^{-7} and the family under test is left out. Without simulated evolution, MRFy achieves an AUC of 0.86, outperforming SMURFLite without simulated evolution (SMURFLite achieved an AUC of 0.77 with an interleave threshold of 2, and 0.81 with an interleave threshold of 4). When simulated evolution is added, MRFy achieves an AUC of 0.92, outperforming SMURFLite with an interleave threshold of 2, but falling just short of the 0.94 AUC SMURFLite demonstrates with an interleave threshold of 4 and simulated evolution. We note that the running

time required for SMURFLite with an interleave threshold of 4 was roughly 10 minutes; MRFy required only 30 seconds to achieve comparable results. The run-time cost of using SMURFLite to produce initial guesses for MRFy is less than 2 seconds; this initial guess does not take advantage of multiple cores, however.

Without using SMURFLite to provide an initial guess, and instead using the template-based model as described in Section 2.2, MRFy outperforms SMURFLite in terms of AUC on four of the β -barrel superfamilies, while SMURFLite outperforms MRFy on three. With SMURFLite initial guesses, MRFy outperformed SMURFLite on all but one superfamily. In particular, we note improvement on the “Nucleic acid-binding proteins” superfamily; see Figure 5 for the results of ROC analysis. In general, MRFy improved the performance on the superfamilies on which SMURFLite performed well, though it did not improve the performance on those superfamilies for which SMURFLite had not performed well in our previous work. Thus, with the exception of the “Barwin-like endoglucanase” superfamily, the added β -strand information does not seem to help MRFy significantly in the cases where HMMER, RAPTOR, CNFPred, or HHPred performed best.

Figure 5: ROC curves and AUC statistic for the “Nucleic acid-binding proteins” superfamily. For this analysis, MRFy was run using the options corresponding to “MRFy²” in Table 4; namely, local search with a time limit of 30 seconds, and template-based initial guesses.

4. DISCUSSION

We have presented MRFy, a method that finds alignments of protein sequences to Markov random field models using stochastic search. MRFy outperforms SMURFLite in most cases, but we should consider several possible enhancements to MRFy that might improve its performance. As demonstrated on the β -sandwich superfamily, MRFy with local search achieves a globally optimal alignment when given 5 minutes of run-time, but fails to find a score close to this when given only 30 seconds. It was not immediately clear how to bring convergence testing into the local search model, but doing so might achieve results comparable to the 5 minute results in less time.

Due to differing cost models, running-time comparisons with the other tools we tested are difficult. For example, RAPTOR and CNFPred both rely on a PSI-BLAST search, which can easily take

15 minutes on a 48-core server. However, the results of this PSI-BLAST search are specific to a query sequence, not to a threading template, and so this running time could be amortized over all the templates to which a query was to be aligned. The alignment phase of a RAPTOR or CNFPred query to a single template takes 1-2 seconds on a 48-core server. In contrast, a single HMMER alignment requires less than 1 second, even though it does not take advantage of multiple cores. HHPred does not align a query to a particular template, but rather aligns a sequence profile or hidden Markov model to each of a set of Pfam databases; such a search typically requires less than 1 second per alignment, but as with RAPTOR and CNFPred, the time required to build a sequence profile from a query sequence dominates the overall running time.

It is also worth noting that unlike MRFy, all the other tools in our comparison are computing an exact solution to some problem; MRFy can benefit from additional running time to find a better local optimum with higher probability. In contrast, SMURFLite always computes the optimal solution to a problem whose complexity can be varied based on the β -strand topology, and HMMER’s run-time complexity is always linear in the product of the length of the model and the length of the query sequence.

We hope that MRFy will be useful for whole-genome annotation of newly-sequenced organisms. The tradeoff of time versus accuracy suggests a two-phase approach to this task: a scan with relatively strict run-time performance requirements (perhaps no more than ten seconds per alignment) coupled with a relatively loose p -value threshold would produce a number of candidates, many of which would likely be false positives. Then, MRFy could be re-run on these candidates with more computationally demanding settings, and with a more strict p -value threshold. MRFy computes p -values identically to SMURFLite: an extreme value distribution [7] is fitted to a distribution of raw scores, and then a p -value is computed as $1 - \text{cdf}(x)$ for any raw MRFy score x . Computing the p -value accurately in the face of different search intensities might require fitting multiple distributions, each for a different level of search intensity. Otherwise, if the distribution is obtained with an intensive search, then at less-intensive search parameters, true positives may result in poor p -values; similarly, if the distribution is obtained with a quick search, then more-intensive search parameters might result in false positives scoring comparatively well, and appearing to have good p -values.

As in [6], we compared MRFy to HHPred [26]. As discussed, HHPred has an advantage in that it builds profiles based on all of protein sequence space. As a future enhancement to MRFy, we plan to introduce query profiles, so that the MRFy alignment is to a sequence profile built from the query sequence, rather than just the query sequence. However, this will introduce a run-time performance hit in two ways. First, the time to run a sequence homology search using the BLAST [1] family of tools can be significant, though the work on compressively-accelerated algorithms by Loh, et al. [18] and Daniels, et al. [4] may reduce this impact. Second, computing the Viterbi and β -pairing scores naively will require time directly proportional to the number of sequences in the query profile. Representing these query sequences as sets of residue frequency vectors should help; there may be other approaches to consider as well.

Thus far, only β -strand interactions lead to non-local interactions in the MRFy Markov random field. In the future, we will investigate fitting other secondary structural elements (the α -helices) into this model. In addition, disulfide bonds, which can occur between cysteine residues and have been shown to be highly conserved [22, 30], would appear to fit easily into this model.

Table 4: AUC on Beta-Barrel superfamilies

	HMMER	RAPTOR	HHPred	RaptorX	CNFPred	SMURFLite	MRfY ¹	MRfY ²	MRfY ³
MRfY performs best									
Translation proteins	-	-	0.66	0.68	0.68	0.93	0.95	0.91	0.95
Barwin-like endoglucanases	-	-	0.75	0.79	0.79	0.77	0.86	0.92	0.94
Tudor/PWWP/MBT	0.78	0.74	0.67	0.83	0.83	0.83	0.86	0.86	0.86
Nucleic acid-binding proteins	0.75	-	0.67	0.66	0.66	0.92	0.75	0.95	0.95
Cyclophilin-like	0.67	0.61	0.7	0.68	0.68	0.85	0.82	0.80	0.85
Sm-like ribonucleoproteins	0.73	0.71	0.77	0.75	0.75	0.85	0.77	0.77	0.87
Prokaryotic SH3-related domain	0.81	-	-	-	-	0.83	0.73	0.72	0.84
HHPred performs best									
Translation proteins SH3-like	0.83	0.81	0.86	0.71	0.71	0.62	-	0.63	0.63
CNFPred performs best									
PDZ domain-like	0.96	1.0	0.99	1.0	1.0	0.97	0.95	0.95	0.96
FMN-binding split barrel	0.62	0.82	0.61	0.93	0.93	-	-	-	-
HMMER performs best									
Electron Transport accessory proteins	0.84	-	0.77	-	-	0.66	-	0.68	0.68

Note: for SMURFLite, value indicated is the best of all values from [6]. Note that RaptorX and CNFPred produced identical AUC scores. MRfY results were using local search with a time limit of 30 seconds. MRfY legend: [1]: MRfY without simulated evolution, with initial guess provided by the template-based model as described in Section 2.2. [2]: MRfY with simulated evolution, and initial guess provided by the template-based model. [3]: MRfY with simulated evolution and initial guess provided by SMURFLite. A dash ('-') in a result entry indicates the method failed on these structures, i.e. an AUC of less than 0.6

5. ACKNOWLEDGEMENTS

Thanks to Richard Senington for the original local search implementation, Jian Peng for useful discussions, and Sheng Wang for providing access to RaptorX and CNFPred for comparison purposes. This work was funded in part by NIH grant 1R01GM080330-01A1 (to L.C.).

References

- [1] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, Sept. 1997.
- [2] H. M. Berman, T. N. Bhat, P. E. Bourne, Z. Feng, G. Gilliland, H. Weissig, and J. Westbrook. The Protein Data Bank and the challenge of structural genomics. *Nature structural biology*, 7 Suppl:957–959, Nov. 2000.
- [3] L. Cowen, P. Bradley, M. Menke, J. King, and B. Berger. Predicting the beta-helix fold from protein sequence data. *Journal of Computational Biology*, 9(2):261–276, 2002.
- [4] N. M. Daniels, A. Gallant, J. Peng, L. J. Cowen, M. Baym, and B. Berger. Compressive genomics for protein databases. *Bioinformatics*, 29(13):i283–i290, July 2013.
- [5] N. M. Daniels, A. Gallant, and N. Ramsey. Experience report: Haskell in computational biology. In *Proceedings of the 17th ACM SIGPLAN international conference on functional programming*, pages 227–234, Sept. 2012.
- [6] N. M. Daniels, R. Hosur, B. Berger, and L. J. Cowen. SMURFLite: combining simplified Markov random fields with simulated evolution improves remote homology detection for beta-structural proteins into the twilight zone. *Bioinformatics*, 28(9):1216–1222, May 2012.
- [7] S. R. Eddy. Profile hidden Markov models. *Bioinformatics*, 14(9):755–763, Oct. 1998.
- [8] V. Gopalakrishnan and P. Weigele. Conditional graphical models for protein structural motif recognition. *Journal of Computational Biology*, 2009.
- [9] P. V. Hentzenryck and L. Michel. *Constraint-based local search*. The MIT Press, Cambridge, MA, 2005.
- [10] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. *ACM SIGART Bulletin*, (63):49, June 1977.
- [11] R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: extension and analysis of the basic method. *Computer applications in the biosciences : CABIOS*, 12(2):95–107, Apr. 1996.
- [12] J. P. S. W. J. X. Jianzhu Ma. A conditional neural fields model for protein threading. *Bioinformatics*, 28(12):i59, June 2012.
- [13] S. Kirkpatrick and M. P. Vecchi. Optimization by simulated annealing. *Science (New York, N.Y.)*, 1983.
- [14] A. Kumar and L. Cowen. Augmented training of hidden Markov models to recognize remote homologs via simulated evolution. *Bioinformatics*, 25(13):1602–1608, 2009.
- [15] A. Kumar and L. Cowen. Recognition of beta-structural motifs using hidden Markov models trained with simulated evolution. *Bioinformatics*, 26(ISMB 2010):i287–i293, 2010.
- [16] R. H. Lathrop and T. F. Smith. Global optimum protein threading with gapped alignment and empirical pair score functions. *Journal of Molecular Biology*, 255(4):641–665, Feb. 1996.
- [17] S. Lifson and C. Sander. Specific recognition in the tertiary structure of [beta]-sheets of proteins. *Journal of Molecular Biology*, 139:627–629, 1980.
- [18] P.-R. Loh, M. Baym, and B. Berger. Compressive genomics. *Nature biotechnology*, 30(7):627–630, July 2012.
- [19] L. J. McGuffin, K. Bryson, and D. T. Jones. The PSIPRED protein structure prediction server. *Bioinformatics*, 2000.
- [20] M. Menke, B. Berger, and L. Cowen. Markov random fields reveal an N-terminal double beta-propeller motif as part of a bacterial hybrid two-component sensor system. *Proceedings of the National Academy of Science*, 2010.
- [21] A. Murzin, S. Brenner, and T. Hubbard. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247(4):536–540, 1995.
- [22] G. Naamati, M. Askenazi, and M. Linial. ClanTox: a classifier of short animal toxins. *Nucleic Acids Research*, 37(Web Server issue):W363–8, July 2009.

- [23] O. Olmea, B. Rost, and A. Valencia. Effective use of sequence correlation and conservation in fold recognition. *Journal of Molecular Biology*, 293(5):1221–1239, 1999.
- [24] J. Peng and J. Xu. A multiple-template approach to protein threading. *Proteins: Structure, Function, and Bioinformatics*, 2011.
- [25] J. Peng and J. Xu. RaptorX: exploiting structure information for protein alignment by statistical inference. *Proteins: Structure, Function, and Bioinformatics*, 2011.
- [26] J. Söding. Protein homology detection by HMM-HMM comparison. *Bioinformatics*, 21(7):951–960, Mar. 2005.
- [27] P. Sonego, A. Kocsor, and S. Pongor. ROC analysis: applications to the classification of biological sequences and 3D structures. *Briefings in bioinformatics*, 9(3):198–209, May 2008.
- [28] R. E. Steward and J. M. Thornton. Prediction of strand pairing in antiparallel and parallel β -sheets using information theory. *Proteins: Structure, Function, and Bioinformatics*, 48:178–191, 2002.
- [29] J. Thomas, N. Ramakrishnan, and C. Bailey-Kellogg. Graphical models of residue coupling in protein families. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, 2008.
- [30] Y. Tirosh, N. Morpurgo, M. Cohen, M. Linial, and G. Bloch. Raalin, a transcript enriched in the honey bee brain, is a remnant of genomic rearrangement in *Hymenoptera*. *Insect molecular biology*, 21(3):305–318, June 2012.
- [31] J. V. White, I. Muchnik, and T. F. Smith. Modeling protein cores with Markov random fields. *Mathematical biosciences*, 124(2):149–179, Dec. 1994.
- [32] J. Xu, M. Li, D. Kim, and Y. Xu. Raptor: Optimal protein threading by linear programming. *Journal of Bioinformatics and Computational Biology*, 1(1):95–117, 2003.
- [33] H. Zhu and W. Braun. Sequence specificity, statistical potentials, and three-dimensional structure prediction with self-correcting distance geometry calculations of beta-sheet formation in proteins. *Protein Science*, 8(2):326–342, 1999.