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

Noah M. Daniels, Andrew Gallant, Norman Ramsey, and Lenore J. Cowen

**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 percent improvement in mean Area Under the Curve for the motif recognition problem as compared to HMMER, 25 percent improvement as compared to RAPTOR, 14 percent improvement as compared to HHPred, and a 18 percent 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://mrfy.cs.tufts.edu/.

**Index Terms**—Protein structure prediction, remote homology detection, structural bioinformatics

## 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 $\beta$-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 [1]. Many of the popular sequence-based methods for recognizing close homologs, such as Profile Hidden Markov Models (HMMs) [2], [3], perform particularly poorly on $\beta$-structural motifs, precisely because the HMM model is not powerful enough to capture these long-range dependencies [4], [5], [6], [7], [8]. Several ways to generalize these HMMs to Markov random fields (MRFs) have therefore been proposed [1], [9], [10], [11], [12], [13], [14], but with the additional power of the random field come 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 et al. introduced SMURF [1], a Markov random field method designed to recognize protein sequences that fold into $\beta$-propeller shapes. While SMURF greatly improved $\beta$-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 $\beta$-sheets, thereby capturing some of the strongest long-range dependencies in $\beta$-structural motifs. The pairwise statistical preferences were learned from solved $\beta$-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 $\beta$-structures. When run on $\beta$-barrels, or other $\beta$-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 $\beta$-strands, namely those that are more local in sequence. This is the methodology we introduced in a 2012 paper of Daniels et al. when we designed SMURFLite [15]. 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 conducted a stringent leave-family-out cross-validation experiment involving recognizing different $\beta$-barrel superfamilies. In this experiment, MRFy, combined with Kumar and Cowen's *simulated evolution* [16], [17] outperforms HMMER [2] (a popular HMM method), Raptor [18]

- *N.M. Daniels is with Mathematics Department and Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139. E-mail: ndaniels@csail.mit.edu.*
- *A. Gallant, N. Ramsey, and L.J. Cowen are with the Department of Computer Science, Tufts University, Medford, MA 02451. E-mail: {agallant, nr}@cs.tufts.edu, lenore.cowen@tufts.edu.*

*Manuscript received 21 Nov. 2013; revised 30 May 2014; accepted 23 July 2014. Date of publication 30 July 2014; date of current version 30 Jan. 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TCBB.2014.2344682*

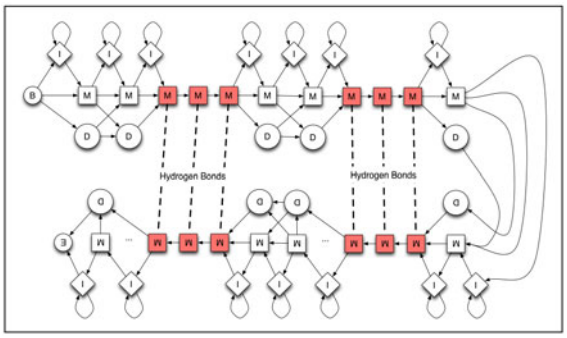


Fig. 1. A MRFy Markov random field with two $\beta$-strand pairs.

(a popular threading method), RaptorX [19] and CNFPred [20] (state-of the art threading methods), and HHPred [21] (a profile-profile HMM method that has performed well at recent CASP competitions). MRFy is available for download at http://mrfy.cs.tufts.edu, under the GNU Public License, version 2. MRFy is written in the Haskell functional programming language; some details of implementation are further discussed in [22].

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 $\beta$-barrel superfamilies considered in [15], we show an improvement of 3.4 percent in the mean (3.6 percent in the median) Area Under Curve (AUC) for $\beta$-structural motif recognition as compared to the SMURFLite results in [15]. For the same benchmark set, MRFy initialized using SMURFLite shows an improvement of 14 percent in the mean (15 percent in the median) over HMMER ([2]), an improvement of 25 percent mean (40 percent median) as compared to RAPTOR ([18]), an improvement of 18 percent mean (21 percent median) over CNFPred [20] and RaptorX [19], and an improvement of 14 percent in the mean (23 percent median) in AUC over HHPred ([21]). We were able to nearly match this performance (within 1 percent in AUC) independent of any use of the SMURFLite codebase, by using multiple different scalings of the size of the template to the test sequences; see Section 2.3 for details. This is significant, because it uses only the MRFy package itself, so it is very easy for others to run. 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.

## 2 Methods

### 2.1 Markov Random Field Model

MRFy builds on the SMURF and SMURFLite Markov random field model [15], which uses multidimensional dynamic programming to simultaneously capture both standard HMM models and the pairwise interactions between amino acid residues bonded together in $\beta$-sheets.

In particular, the "Plan7" hidden Markov model as implemented by HMMER (so named because exactly seven transition edges are allowed per node) is modified to represent hydrogen-bonded $\beta$-strands with additional, non-local edges. Because the $\beta$-strands in a SMURF or MRFy template represent *consensus* $\beta$-strands, those present in at least some fraction (in our experiments, at least half) of the sequences participating in the training alignment, we prohibit insertions and deletions in those strands. Thus, we collapse those nodes of the "Plan7" model to be just match states; the transitions to insertion and deletion states are removed. Fig. 1 illustrates this architecture.

The standard form of the Viterbi recurrence relations for computing the most likely path of a sequence through a hidden Markov model, as incorporated in HMMER [2] is

$$V_j^M(i) = \frac{e_{M_j}(x_i)}{q_{x_i}} \times \max \begin{cases} V_{j-1}^M(i-1) \times a_{M_{j-1}M_j} \\ V_{j-1}^I(i-1) \times a_{I_{j-1}M_j} \\ V_{j-1}^D(i-1) \times a_{D_{j-1}M_j} \end{cases}$$

$$V_j^I(i) = \frac{e_{I_j}(x_i)}{q_{x_i}} \times \max \begin{cases} V_j^M(i-1) \times a_{M_jI_j} \\ V_j^I(i-1) \times a_{I_jI_j} \end{cases}$$

$$V_j^D(i) = \max \begin{cases} V_{j-1}^M(i) \times a_{M_{j-1}D_j} \\ V_{j-1}^D(i) \times a_{D_{j-1}D_j}. \end{cases}$$

In the SMURF or MRFy Markov random field model, we add non-local interactions to these probabilities, resulting in conditional probabilities. When column $j$ of an alignment is part of a $\beta$-strand and is paired with another column $\pi(j)$, the probability of finding amino acid $x_i$ in column $j$ depends on whatever amino acid $x'$ is in column $\pi(i)$. If $x'$ is in position $i'$ in the query sequence, Viterbi's equations are altered; for example, $V_j'^M(i)$ depends not only on $V_{j-1}'^M(i-1)$ but also on $V_{\pi(j)}'^M(i')$. The distance between $j$ and $\pi(j)$ can be as small as a few columns or as large as a few hundreds of columns. Because $V_j'^M(i)$ depends not only on nearby values but also on $V_{\pi(j)}'^M(i')$, we must modify the Viterbi recurrence relations.

Note that hydrogen-bonded $\beta$-strand residues may only occupy match states in the Markov random field, so only the corresponding terms of the recurrence relation need be modified. The revised Viterbi recurrence relation for the Markov random field is

$$V_j^M(i) = \frac{e_{M_j}(x_i)}{q_{x_i}} \times \max \begin{cases} V_{j-1}^M(i-1) \times a_{M_{j-1}M_j} \times P(x_i \,|\, x_{\pi j}) \\ V_{j-1}^I(i-1) \times a_{I_{j-1}M_j} \times P(x_i \,|\, x_{\pi j}) \\ V_{j-1}^D(i-1) \times a_{D_{j-1}M_j} \times P(x_i \,|\, x_{\pi j}), \end{cases}$$

where $x_{\pi j}$ represents the amino acid in column $\pi j$, which is hydrogen-bonded to the amino acid $x_i$ in column $j$. Then, transforming to negative log space, we actually solve

$$V_j'^M(i) = e'_{M_j}(x_i) + \min \begin{cases} a'_{M_{j-1}M_j} + V_{j-1}'^M(i-1) + P'(x_i \,|\, x_{\pi j}) \\ a'_{I_{j-1}M_j} + V_{j-1}'^I(i-1) + P'(x_i \,|\, x_{\pi j}) \\ a'_{D_{j-1}M_j} + V_{j-1}'^D(i-1) + P'(x_i \,|\, x_{\pi j}) \end{cases}$$

$$V_j'^I(i) = e'_{I_j}(x_i) + \min \begin{cases} a'_{M_jI_j} + V_j'^M(i-1) \\ a'_{I_jI_j} + V_j'^I(i-1) \end{cases}$$

$$V_j'^D(i) = \min \begin{cases} a'_{M_{j-1}D_j} + V_{j-1}'^M(i) \\ a'_{D_{j-1}D_j} + V_{j-1}'^D(i) \end{cases}$$

given the transformations:

$$a'_{s\hat{s}} = -\log a_{s\hat{s}}$$
$$e'_s(x) = -\log \frac{e_s(x)}{q_x}$$
$$V'^M_j(i) = -\log V^M_j(i)$$
$$P'(x_i \mid x_{\pi j}) = -\log P(x_i \mid x_{\pi j}).$$

This is exactly the recurrence relation that SMURF [1] and SMURFLite [15] solve using multidimensional dynamic programming. Note that gaps are prohibited within $\beta$-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 $\beta$ 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 $\beta$-strands is defined as the number of other $\beta$-strands that occur between those two strands in the protein sequence, plus one. As demonstrated by Daniels et al. [15], as the maximum interleave of a pair of matched $\beta$-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 $\beta$-strand can be thought of as breaking the larger model into two smaller models; collectively, all the $\beta$-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 $\beta$-strands.

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

$$V'^M_j(i) = e'_{M_j}(x_i) + V'^M_{j-1}(i-1) + a'_{M_{j-1}M_j} + P(x_i \mid x_{\pi j})$$

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 $\beta$-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 $\beta$-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 $\beta$-strand, computing the score for each one, and choose the best-scoring placement. Note that parallel versus anti-parallel $\beta$-strands are captured by the structural alignment from which the SMURF or MRFy Markov random field is built, and thus need not be treated any differently.

Metaphorically, we can picture the residues of the query sequence as beads, and the Markov random field as the string of a necklace. The $\beta$-strands can be thought of as particular substrings of the string that must be covered by beads, while non-$\beta$ regions may be exposed (resulting in delete states in the model). To continue the metaphor, we may jam extra beads onto non-$\beta$ 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 $\beta$-regions are covered. Since the regions between $\beta$-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 $\beta$-strand nodes.

### 2.2 Proof that There are an Exponential Number of Assignments

However, it is easy to show that there are an exponential number of such assignments, in terms of the length of the query sequence.

**Definition** *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 $\beta$-strands $b_i, i \in (1..k)$. Each $\beta$-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 $r_i, i \in (1..n)$.*

**Definition** *Let* $L = \sum_{i,i<=k} l_i$.

**Lemma 2.1.** *Given a model $(N, B)$ and a query sequence $R$, $L$ residues are placed in $\beta$-strands.*

**Proof.** Because each $\beta$-strand $b_i$ must be populated by exactly $l_i$ residues, $\forall j, j > 1$, $b_{ij}$ is uniquely determined by the sequence $R$. For each $\beta$-strand position $b_{ij}$, one residue is placed. Thus, $\sum_{i,i<=k} l_i$ residues are placed in $\beta$-strands. □

**Theorem 2.2.** *For a Markov random field $(N, B)$ with $k$ $\beta$-strands $b_i$, each of length $l_i$, and thus containing positions for residues $b_{ij}$ and a query sequence $ri$ of length $n$, there are $\Theta(n^k)$ ways to assign residues to the $\beta$-strands.*

**Proof.** From the $n$ residues in the query sequence $R$, we need to place $L$ residues across all $B$ $\beta$-strands. We represent this as choosing an index $i \in (1..n)$ for the first position $b_{i1}$ of each $\beta$-strand. Since each $\beta$-strand $b_i$ consumes $l_i$ residues, this choice for the first $\beta$-strand, $b_{11}$, leaves $n - L - l_i$ possible placements for $b_{21}$. In practice, $\beta$-strands range from two to twelve residues, so to simplify counting, we assume each $l_i$ is simply a maximum length $l_{max}$. This only decreases the number of possible assignments, yielding a lower bound on the number of placements. Then choosing an index to place on $b_{i1}$, in general, leaves $n - L - (i \times l_{max})$ choices for $b_{(i+1)1}$. Thus, there are

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

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

A typical Markov random field might have 10 or 20 $\beta$-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. Clearly, this computation can be broken into many parallel parts, but this still poses an intractable problem in many cases.

### 2.3 Stochastic Search

Since an exhaustive search for an optimal alignment of a protein sequence to our Markov random field model is computationally impractical in many real cases, we turn to stochastic search to mitigate this complexity.

Stochastic search encompasses a family of approaches for finding optimal or near-optimal solutions to optimization problems. Stochastic search approaches are promising when a search space is large, so that exhaustive search is prohibitive, and when an optimization problem does not lend itself to analytic solutions. The generic form of stochastic search is that a solution is guessed at and evaluated, and then subsequent guesses are made as refinements to this initial guess, until some termination condition is met. The function used for evaluation is called the *objective function*.

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 [1] finds this minimum exactly, using multi-dimensional 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 $\beta$-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 (five 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.

Many stochastic search techniques rely on a gradient ascent (or descent) approach, which makes moves (or refines guesses) along the steepest gradient, leading quickly to local optima; various heuristics such as simulated annealing [23] can then help avoid getting stuck in poor local optima.

However, we know of no way to compute a gradient on the search space of $\beta$-strand placements, and so we must take approaches that do not rely on this gradient. Instead, we must rely on a random-mutation model of search, which generates one or more candidate solutions (guesses) from a previous solution, and then evaluates the cost function (in our case, the augmented Viterbi score) to determine whether those guesses are better or worse than the previous step. This can be likened to climbing a hill in the dark, feeling one's way with one foot before committing to a step. This approach is referred to as *random-mutation hill climbing* [24].

In our representation, a particular solution is represented by an ordered list of integers, one integer per $\beta$-strand in the Markov random field. The value of each integer indicates the index, in the query sequence, of the residue assigned to the first position of that $\beta$-strand. Since the alignments to the regions of the Markov random field are solved exactly by the Viterbi algorithm, this ordered list of integers uniquely represents a solution to a Markov random field.

While the picture we have presented for our Markov random field model is most precisely explained by assigning residue indices to the positions of $\beta$-strands, it may be more intuitive to consider the equivalent problem of "sliding" these $\beta$-strands along the query sequence. We will use this analogy in the following description of initial guesses.

We explored five models for generating initial guesses for our search techniques; four are internal to MRFy, and one is a hybrid approach that relies on SMURFLite. A *random-placement* model uniformly positions the $\beta$-strands along the query sequence, under the constraint that only legal placements may be generated, and thus the placement of any $\beta$-strand must leave room for all the other $\beta$-strands in the model.

A *secondary-structure* prediction model uses the PSIPRED [25] secondary-structure prediction program to determine the positions of $\beta$-strands. Given a PSIPRED prediction for the secondary structure of a query sequence, we place $\beta$-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 $\beta$-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 $\beta$-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 $\beta$-strands in scaled positions. Note that we do not scale the $\beta$-strands themselves; their lengths are preserved. We scale only the distances between $\beta$-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 heterogenous solutions. *This is the initial guess method that we used to generate the results in Tables 1, 2, and 3 in Fig. 5, and the MRFy[1] and MRFy[2] results in Table 4.*

Next, we implemented a variant of the template-based model that attempts to address an observed weakness of that model. Suppose a query sequence actually contains several domains, and only one of those domains is actually homologous to a particular template. Then, the template-based initial guess will be a poor initial guess, as the $\beta$-strands will be distributed along the entire multi-domain sequence, rather than concentrated in the homologous region. Thus, it will take the stochastic search a long time to converge on an optimal placement. This *multi-scaled template-based model* scales the $\beta$-strand placements of the template up to a variety of possible corresponding lengths in the query sequence. Assume the model from the template $T$ is of length $t$, and let the length of the query sequence $Q$ be $q$. Then, if $q > t$, consider possible lengths $r$ of the region $R$ of the $Q$ homologous to $T$, ranging from $t$ up to $q$, in increments of 10. Also, consider that the starting position of each of these regions can range from 0 up to $q - r$, also in

TABLE 1
Stochastic Search Performance on Eight-Bladed $\beta$-Propeller

| | Min Score | Mean Score | Std Score | Mean Time (s) |
|---|---|---|---|---|
| SA 200 | 2,112 | 2,139 | 12.2 | 29.3 |
| SA 500 | 2,129 | 2,146 | 9.3 | 1,020 |
| SA 1000 | 2,112 | 2,130 | 7.8 | 3,314 |
| GA 1000/10 | 2,105 | 2,126 | 6.6 | 285 |
| GA 1000/50 | 2,094 | 2,118 | 7.7 | 1,239 |
| GA 1000/100 | 2,107 | 2,120 | **3.8** | 548 |
| GA 10000/10 | 2,087 | 2,111 | 7.2 | 5,809 |
| GA 10000/50 | 2,094 | 2,112 | 7.1 | 5,174 |
| GA 10000/100 | 2,079 | 2,114 | 9.0 | 10,226 |
| LS 10 s | 1,992 | 2,015 | 19.4 | **10** |
| LS 30 s | 1,982 | 1,991 | 10.9 | 30 |
| LS 5 m | **1,818** | **1,876** | 37.2 | 300 |

*Performance of stochastic search techniques on an 8-bladed $\beta$-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 1,000 generations. For the Genetic Algorithm, we show results for convergence thresholds of 10, 50, and 100 generations, and for population sizes of 1,000 and 10,000. 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 1,781, 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 2.3. These results show that local search outperforms the other search strategies, and presents the tradeoff between the running time and accuracy of stochastic search.*

increments of 10. Each of these placements is an initial guess for an instance of the stochastic search. Note that this initial guess model only differs from the template-based initial guess when $q$ is significantly larger than $t$, and it is designed to aid in detecting individual homologous regions (possibly domains) within a query sequence that is *not* presumed to contain only a single domain. *This is the initial guess method that we used to generate the results for MRFy[3] 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 [15], from which we have removed any $\beta$-strand pairs that exceed an interleave threshold of 2 (that is, any pairs that have more than two other $\beta$-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 $\beta$-strands that were removed. In order to do so, we greedily move the placement of each $\beta$-strand in $a'$ as needed to make room for those other $\beta$-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 MRFy[4] 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 Fig. 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.

#### 2.3.1 Simulated Annealing

Simulated annealing [23] is a heuristic for stochastic search, inspired by the physical process of annealing in metals. Whereas a simple hill-climbing approach will always

TABLE 2
Stochastic Search Performance on "Barwin-Like" $\beta$-Barrel

| | Min Score | Mean Score | Std Score | Mean Time (s) | Optimal |
|---|---|---|---|---|---|
| SA 200 | 1,064 | 1,071 | 3.8 | 79.5 | 0 |
| SA 500 | 1,047 | 1,063 | 7.6 | 104 | 0 |
| SA 1000 | 1,024 | 1,047 | 14.0 | 523 | 0 |
| GA 1000/10 | 1,061 | 1,069 | 3.6 | 232 | 0 |
| GA 1000/50 | 1,059 | 1,066 | 3.1 | 442 | 0 |
| GA 1000/100 | 1,058 | 1,069 | 4.0 | 1,382 | 0 |
| GA 10000/10 | 1,058 | 1,063 | 2.5 | 8,205 | 0 |
| GA 10000/50 | 1,059 | 1,061 | **2.2** | 10,306 | 0 |
| GA 10000/100 | 1,057 | 1,061 | **2.2** | 16,395 | 0 |
| LS 10 s | **978** | 995 | 16.2 | **10** | 0.1 |
| LS 30 s | **978** | 987 | 6.9 | 30 | 0.2 |
| LS 5 m | **978** | **981** | 2.9 | 300 | 0.4 |

*Performance of stochastic search techniques on the "Barwin-like endoglucanases" $\beta$-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 1,000 generations. For the Genetic Algorithm, we show results for convergence thresholds of 10, 50, and 100 generations, and for population sizes of 1,000 and 10,000. 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 2.3. 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 $\beta$-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 | 1,869 | 0 |
| GA 1000/50 | 878 | 883 | **2.5** | 1,305 | 0 |
| GA 1000/100 | 865 | 878 | 5.6 | 4,309 | 0 |
| GA 10000/10 | 872 | 877 | **2.5** | 6,999 | 0 |
| GA 10000/50 | 875 | 879 | 3.1 | 5,317 | 0 |
| GA 10000/100 | 869 | 875 | 4.5 | 10,733 | 0 |
| LS 10 s | 771 | 826 | 31.7 | **10** | 0 |
| LS 30 s | 740 | 791 | 47.0 | 30 | 0 |
| LS 5 m | **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 1,000 generations. For the Genetic Algorithm, we show results for convergence thresholds of 10, 50, and 100 generations, and for population sizes of 1,000 and 10,000. 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 2.3. These results show that local search outperforms the other search strategies, and presents the tradeoff between the running time and accuracy of stochastic search.*

terminate at the first local optimum encountered, simulated annealing introduces an *acceptance probability function*

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

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 progresses, 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 $\beta$-strand placements are determined by $s$.

We implemented simulated annealing in MRFy according to this model. We also implemented a *multi-start* version of

TABLE 4
AUC on Beta-Barrel Superfamilies

| | HMMER | RAPTOR | HHPred | RaptorX | CNFPred | SMURFLite | MRFy[1] | MRFy[2] | MRFy[3] | MRFy[4] |
|---|---|---|---|---|---|---|---|---|---|---|
| **MRFy performs best** | | | | | | | | | | |
| Translation proteins | - | - | 0.66 | 0.68 | 0.68 | 0.93 | **0.95** | 0.91 | **0.95** | **0.95** |
| Barwin-like endoglucanases | - | - | 0.75 | 0.79 | 0.79 | 0.77 | 0.86 | 0.92 | 0.93 | **0.94** |
| Tudor/PWWP/MBT | 0.78 | 0.74 | 0.67 | 0.83 | 0.83 | 0.83 | **0.86** | **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** | **0.95** |
| Cyclophilin-like | 0.67 | 0.61 | 0.7 | 0.68 | 0.68 | **0.85** | 0.82 | 0.80 | **0.85** | **0.85** |
| Sm-like ribonucleoproteins | 0.73 | 0.71 | 0.77 | 0.75 | 0.75 | 0.85 | 0.77 | 0.77 | 0.85 | **0.87** |
| Prokaryotic SH3-related domain | 0.81 | - | - | - | - | 0.83 | 0.73 | 0.72 | **0.84** | **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 | 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 | 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 | 0.68 |

*Note: For SMURFLite, value indicated is the best of all values from [15]. 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 2.3. [2]: MRFy with simulated evolution, and initial guess provided by the template-based model. [3]: MRFy with simulated evolution and initial guess provided by the multi-scaled template-based model. [4]: 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.*

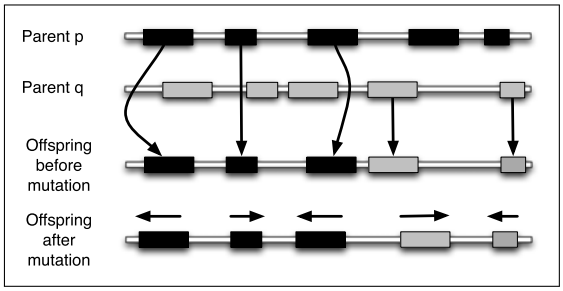


Fig. 2. The crossover and mutation process in MRFy's genetic algorithm implementation. Given parent $p$ (black) and parent $q$ (gray), alternate left and right placements from $p$ and $q$. Then, apply small random mutations to the resulting placement $p'$.

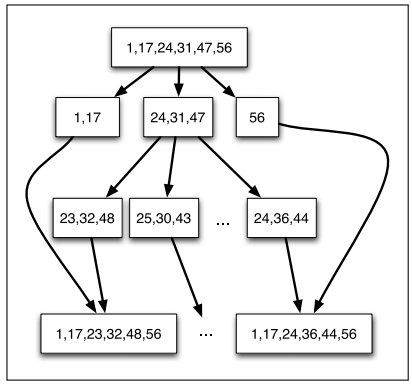


Fig. 3. The diversification step in local search. In this example, $\beta$-strand placements in the middle of the query sequence, corresponding to residue positions 24, 31, and 47, are varied at random within the bounds of the surrounding placements (corresponding to residue positions 17 and 56) in order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

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.

### 2.3.2 Genetic Algorithm

A genetic algorithm [26] 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 $k$ $\beta$-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.
1) 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 (in terms of sequence) $\beta$-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 solution with one "half" of the other. See Fig. 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.

### 2.3.3 Local Search

Constraint-based local search [27] 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 $\beta$-strands cannot overlap, and every residue must be placed in a $\beta$-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 Fig. 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 $\beta$-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, intensification 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 Fig. 5.*

### 2.4 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 "eight-bladed Beta-Propellers", and the superfamilies "Barwin-like endoglucanases" (a $\beta$-barrel superfamily) and "Concanavalin A-like lectins/glucanases" (a $\beta$-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 1,000 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 1,000 and 10,000, 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 32 GB RAM, devoting all 12 cores to MRFy. For each test, we report statistics based on 10 runs for each set of parameters.

### 2.5 Simulated Evolution

In MRFy, we incorporated precisely the same "simulated evolution" implementation, as first proposed by Kumar and Cowen [16], [17], as we did for SMURFLite in [15]. We added pairwise mutations based on $\beta$-strand pairings. We use the same mutation frequencies as in [15]. For each training sequence, we generate 150 new artificial training sequences. For each of these artificial sequences, we mutate at a 50 percent mutation rate per length of the $\beta$-strands. The goal of this approach is to partially compensate for limited training data, particularly in superfamilies that contain relatively few distinct sequences.

### 2.6 Datasets

From SCOP ([28]) version 1.75, we chose the same $\beta$-barrel superfamiles as [15]. 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).

### 2.7 Training and Testing Process

For the $\beta$-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 [15]. 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 ([29]) database with non-redundancy set to a BLAST E-value of $10^{-7}$.

We used MRFy's local search mode (see Section 2.3.3) 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 [30].

## 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 2.4. 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 eight-bladed propellers, the Barwin-like endoglucanase $\beta$-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 eight-bladed propeller fold, SMURF itself returns the best-scoring alignment. The Barwin-like endoglucanase $\beta$-barrel superfamily has an unusually low interleave number for a beta-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 $\beta$-strands, we were able to exhaustively enumerate every possible placement of $\beta$-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 eight-bladed $\beta$-propeller fold, using the *template-based* model for initial guesses described in Section 2.3. 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. Multi-start 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 2,112, in contrast to a global optimum of 1,781, a ratio of 0.843, though it converged in all cases.

In contrast, local search, given 30 seconds, achieves a minimum score of 1,982 (a ratio of 0.899), and even in only 10 seconds achieves a minimum score of 1,992 (a ratio of 0.894). However, the global minimum score of 1,781, which is achieved by SMURF on the 8-bladed $\beta$-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" $\beta$-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 [15] 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 10 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 10 runs.

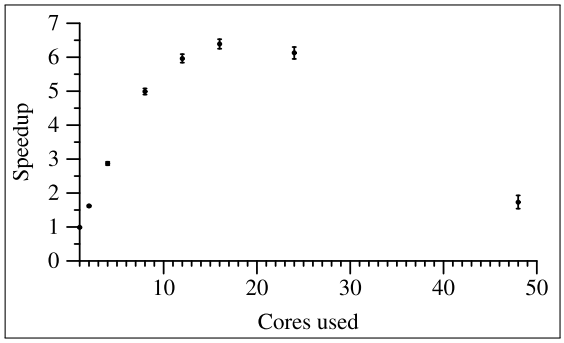


Fig. 4. MRFy's parallel speedup on an eight-bladed $\beta$-propeller, using a 48-core system. After about 12 cores, speedup falls off.

Table 3 indicates the performance of the stochastic search techniques on the "Concanavalin A-like lectins/glucanases" $\beta$-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 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) 10 out of 10 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 $\beta$-propeller" template. The model has 343 nodes, of which 178 appear in 40 $\beta$-strands. The segments between $\beta$-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 $\beta$-strands independently, this benchmark has a lot of parallelism.

Fig. 4 shows speedups when using from 1 to 48 of the cores on a 48-core, 2.3 GHz AMD Opteron 6176 system. Errors are estimated from five runs. After about 12 cores, where MRFy runs six times as fast as sequential code, speedup rolls off. We note that by running four instances of MRFy in parallel on different searches, we would expect to be able to use all 48 cores with about 50 percent 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 $\beta$-strand placements, is scored in parallel). Thus, we would anticipate somewhat better parallel efficiency with a larger number of $\beta$-strands, and worse parallel efficiency with few $\beta$-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.

### 3.2 Remote Homology Detection Accuracy

We compared MRFy using local search and a 30-second time limit, against HMMER [2], Raptor [18], HHPred [21], RaptorX [19], and CNFPred [20]. We performed cross-validation testing on 11 $\beta$-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.

We compared MRFy's performance, both with and without simulated evolution, to the results from [15], for exactly the same set of 11 SCOP $\beta$-barrel superfamilies that were considered. 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, BoostThreader), 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 for the "Barwin-like endoglucanases" superfamily highlighted in [15], SMURFLite performed better as the interleave threshold was increased on this superfamily. Since MRFy discards no $\beta$-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 four training sequences and as few as three 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.

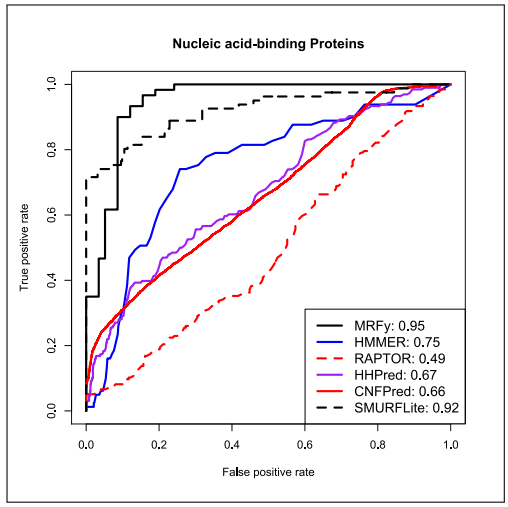


Fig. 5. ROC curves and AUC statistic for the "Nucleic acid-binding proteins" superfamily. For this analysis, MRFy was run using the options corresponding to "MRFy[3]" in Table 4; namely, local search with a time limit of 30 seconds, and multi-scaled template-based initial guesses.

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 2.3, MRFy outperforms SMURFLite in terms of AUC on four of the $\beta$-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 Fig. 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 $\beta$-strand information does not seem to help MRFy significantly in the cases where HMMER, RAPTOR, CNFPred, or HHPred performed best.

### 3.3 Alignment Quality

MRFy, like SMURFLite [15] and HMMER [2], aligns a query sequence to a template that was itself produced from a multiple structure alignment. Beyond using MRFy's alignment score for homology detection, one might consider how well this alignment to a structural template matches the alignment that would result from structurally aligning the query protein to the structural template. We compared MRFy using local search and a 90-second time limit against HMMER and SMURFLite on the same 11 $\beta$-barrel superfamilies as in the previous section. For each superfamily, we left out each of its families in turn, and for each structure in the family left out, we used Matt [31] to re-align that structure to the structural alignment built from the remaining families in the superfamily. For MRFy, SMURFLite, and

TABLE 5
Alignment Quality on Beta-Barrel Superfamilies

| | HMMER | SMURFLite | MRFy[1] | MRFy[2] | MRFy[3] | MRFy[4] |
|---|---|---|---|---|---|---|
| Translation proteins | 46 | 72 | 70 | 71 | **73** | **73** |
| Barwin-like endoglucanases | 53 | 58 | 63 | **65** | 64 | 63 |
| Tudor/PWWP/MBT | 64 | 67 | 66 | **68** | 67 | 67 |
| Nucleic acid-binding proteins | 63 | 71 | 73 | 75 | **75** | 74 |
| Cyclophilin-like | 47 | 62 | 64 | 63 | **66** | 65 |
| Sm-like ribonucleoproteins | 63 | 61 | 67 | 65 | **68** | **68** |
| Prokaryotic SH3-related domain | 71 | 73 | 72 | 74 | **75** | **75** |
| Translation proteins SH3-like | **62** | 51 | 56 | 51 | 50 | 52 |
| PDZ domain-like | 83 | 81 | 80 | 81 | **84** | **84** |
| FMN-binding split barrel | 41 | 43 | 44 | **46** | 44 | 44 |
| Electron Transport accessory proteins | **59** | 48 | 46 | 44 | 47 | 46 |

*Results shown are the percentage of columns in the alignment that are correct with respect to a purely structural alignment performed using Matt. 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 2.3. [2]: MRFy with simulated evolution, and initial guess provided by the template-based model. [3]: MRFy with simulated evolution and initial guess provided by the multi-scaled template-based model. [4]: MRFy with simulated evolution and initial guess provided by SMURFLite.*

HMMER, we then aligned the sequence of each left-out protein to the trained model built from that same superfamily structural alignment. For each such alignment, we counted the number of columns in which the MRFy, SMURFLite, or HMMER alignment was in agreement with the Matt structural alignment. We present the mean of this percentage of correct columns for each $\beta$-barrel superfamily in Table 5.

We note that a direct alignment-quality comparison to RAPTOR, HHPred, and RaptorX is not meaningful, as these other tools do not align a single query sequence to a template derived from a multiple alignment.

In a majority (6 of 11 superfamilies), MRFy with simulated evolution and an initial guess provided by the multi-scaled template-based model performed best, and its alignment was always strictly better than SMURFLite's. This is unsurprising, because MRFy can consider $\beta$-strands that SMURFLite cannot.

## 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, both in terms of homology detection performance and alignment accuracy, but we should consider several possible enhancements to MRFy that might improve its performance. As demonstrated on the $\beta$-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, RaptorX 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 RaptorX 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 RaptorX 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 $\beta$-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.

As we saw with SMURFLite [15], MRFy would be at a disadvantage on superfamilies that contained very few homologous proteins. However, as with SMURFLite, the inclusion of "simulated evolution" somewhat mitigates this concern.

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 [2] is fitted to a distribution of raw scores, and then a $p$-value is computed as $1 - cdf(x)$ 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 [15], we compared MRFy to HHPred [21]. 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 [32] family of tools can be significant, though the work on compressively-accelerated algorithms by Loh et al. [33] and Daniels et al. [34] may reduce this impact. Second, computing the Viterbi and $\beta$-pairing scores naïvely 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.

One limitation of MRFy's approach is that the training phase requires a template to be built from a set of structurally consistent protein chains. However, above the superfamily level of SCOP, while some folds are structurally consistent, particularly the $\beta$-propellers [1], other folds do not align well structurally [35]. We see this even in superfamilies whose alignments do not preserve consensus $\beta$-strands; as pointed out in [35], the 'Translation Proteins SH3-Like Domain,' in which HMMER and HHPred both outperform MRFy, has four $\beta$-strands, but the consensus alignment preserves none of them, eliminating any advantage from the Markov random field.

In order to use MRFy to predict homologs to sets of protein structures that do not align well, it may be useful to separate these superfamilies or folds into more easily-aligned subsets. Additionally, it is not clear for every fold that all constituent superfamilies are homologous, so the task of identifying homologous proteins from different superfamilies but the same fold is more difficult in general. This suggests further investigation into modifying the MRFy approach to be more effective in the detection of truly remote homologs.

Thus far, only $\beta$-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 $\alpha$-helices) into this model. Currently, $\alpha$-helix residues are treated like any other residue; while some of the $\beta$-barrel superfamilies on which MRFy performs well do include $\alpha$-helices, they are not modeled structurally or identified in the alignment. In addition, disulfide bonds, which can occur between cysteine residues and have been shown to be highly conserved [36], [37], would appear to fit easily into this model.

## Acknowledgments

The authors thank Richard Senington for the original local search implementation, Jian Peng for useful discussions, David Fish for graphics assistance, and Sheng Wang for providing access to RaptorX and CNFPred for comparison purposes. This work primarily occured while Noah Daniels was still a Ph.D. student at Tufts, and was funded in part by NIH grant 1R01GM080330-01A1 (to L.C.). Lenore J. Cowen is the corresponding author.

## References

[1] 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," *Proc. Nat. Acad. Sci. USA*, vol. 107, no. 9, pp. 4069–4074, 2010.

[2] S. R. Eddy, "Profile hidden Markov models," *Bioinformatics*, vol. 14, no. 9, pp. 755–763, Oct. 1998.

[3] R. Hughey and A. Krogh, "Hidden Markov models for sequence analysis: Extension and analysis of the basic method," *Comput. Appl. Biosci.: CABIOS*, vol. 12, no. 2, pp. 95–107, Apr. 1996.

[4] S. Lifson and C. Sander, "Specific recognition in the tertiary structure of [beta]-sheets of proteins," *J. Mol. Biol.*, vol. 139, pp. 627–629, 1980.

[5] 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 Sci.*, vol. 8, no. 2, pp. 326–342, 1999.

[6] O. Olmea, B. Rost, and A. Valencia, "Effective use of sequence correlation and conservation in fold recognition," *J. Mol. Biol.*, vol. 293, no. 5, pp. 1221–1239, 1999.

[7] L. Cowen, P. Bradley, M. Menke, J. King, and B. Berger, "Predicting the beta-helix fold from protein sequence data," *J. Comput. Biol.*, vol. 9, no. 2, pp. 261–276, 2002.

[8] R. E. Steward and J. M. Thorton, "Prediction of strand pairing in antiparallel and parallel $\beta$-sheets using information theory," *Proteins: Structure, Function, Bioinformat.*, vol. 48, pp. 178–191, 2002.

[9] J. V. White, I. Muchnik, and T. F. Smith, "Modeling protein cores with Markov random fields," *Math. Biosci.*, vol. 124, no. 2, pp. 149–179, Dec. 1994.

[10] R. H. Lathrop and T. F. Smith, "Global optimum protein threading with gapped alignment and empirical pair score functions," *J. Mol. Biol.*, vol. 255, no. 4, pp. 641–665, Feb. 1996.

[11] J. Thomas, N. Ramakrishnan, and C. Bailey-Kellogg, "Graphical models of residue coupling in protein families," *IEEE/ACM Trans. Comput. Biol. Bioinformat.*, vol. 5, no. 2, pp. 183–197, Apr. 2008.

[12] V. Gopalakrishnan and P. Weigele, "Conditional graphical models for protein structural motif recognition," *J. Comput. Biol.*, vol. 16, no. 5, pp. 639–657, 2009.

[13] J. Peng and J. Xu, "A multiple template approach to protein threading," *Proteins: Structure, Function, Bioinformat.*, vol. 79, no. 6, pp. 1930–1939, 2011.

[14] J. Ma, S. Wang, Z. Wang, and J. Xu, "MRFalign: Protein homology detection through alignment of Markov random fields," *PLoS Comput Biol*, vol. 10, no. 3, p. e1003500, Mar. 2014.

[15] 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*, vol. 28, no. 9, pp. 1216–1222, May 2012.

[16] A. Kumar and L. Cowen, "Augmented training of hidden Markov models to recognize remote homologs via simulated evolution," *Bioinformatics*, vol. 25, no. 13, pp. 1602–1608, 2009.

[17] A. Kumar and L. Cowen, "Recognition of beta-structural motifs using hidden Markov models trained with simulated evolution," *Bioinformatics*, vol. 26, no. 12, pp. i287–i293, 2010.

[18] J. Xu, M. Li, D. Kim, and Y. Xu, "Raptor: Optimal protein threading by linear programming," *J. Bioinformat. Comput. Biol.*, vol. 1, no. 1, pp. 95–117, 2003.

[19] J. Peng and J. Xu, "RaptorX: Exploiting structure information for protein alignment by statistical inference," *Proteins: Structure, Function, Bioinformat.*, vol. 79, Suppl. 10, pp. 161–171, 2011.

[20] J. Ma, J. Peng, S. Wang, and J. Xu, "A conditional neural fields model for protein threading," *Bioinformatics*, vol. 28, no. 12, p. i59, Jun. 2012.

[21] J. Söding, "Protein homology detection by HMM-HMM comparison," *Bioinformatics*, vol. 21, no. 7, pp. 951–960, Mar. 2005.

[22] N. M. Daniels, A. Gallant, and N. Ramsey, "Experience report: Haskell in computational biology," in *Proc. 17th ACM SIGPLAN Int. Conf. Functional Programm.*, Sept. 2012, pp. 227–234.

[23] M. Menke, B. Berger, and L. Cowen, "Matt: local flexibility aids protein multiple structure alignment," *PLoS computational biology 4.1:e10*, 2008.

[24] L. Davis, "Bit-climbing, representational bias, and test suite design," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 18–23.

[25] L. J. McGuffin, K. Bryson, and D. T. Jones, "The PSIPRED protein structure prediction server," *Bioinformatics*, vol. 16, no. 4, pp. 404–405, 2000.
[26] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," *ACM SIGART Bulletin*, no. 63, p. 49, Jun. 1977.
[27] P. V. Hentenryck and L. Michel, *Constraint-Based Local Search*, Cambridge, MA, USA: MIT Press, 2005.
[28] A. Murzin, S. Brenner, and T. Hubbard, "SCOP: A structural classification of proteins database for the investigation of sequences and structures," *J. Mol. Biol.*, vol. 247, no. 4, pp. 536–540, 1995.
[29] 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," *Nat. Struct. Biol.*, vol. 7, pp. 957–959, Nov. 2000.
[30] P. Sonego, A. Kocsor, and S. Pongor, "ROC analysis: Applications to the classification of biological sequences and 3D structures," *Briefings Bioinformat.*, vol. 9, no. 3, pp. 198–209, May 2008.
[31] M. Menke, B. Berger, and L. Cowen, "Matt: Local flexibility aids protein multiple structure alignment," *PLoS Comput. Biol.*, vol. 4, p. e10, 2008.
[32] 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 Res.*, vol. 25, no. 17, pp. 3389–3402, Sep. 1997.
[33] P.-R. Loh, M. Baym, and B. Berger, "Compressive genomics," *Nat. Biotechnol.*, vol. 30, no. 7, pp. 627–630, Jul. 2012.
[34] N. M. Daniels, A. Gallant, J. Peng, L. J. Cowen, M. Baym, and B. Berger, "Compressive genomics for protein databases," *Bioinformatics*, vol. 29, no. 13, pp. i283–i290, Jul. 2013.
[35] N. Daniels, A. Kumar, L. Cowen, and M. Menke, "Touring protein space with matt," *Bioinformat. Res. Appl.*, vol. 6053/2010, pp. 18–28, 2010.
[36] G. Naamati, M. Askenazi, and M. Linial, "ClanTox: A classifier of short animal toxins," *Nucleic Acids Res.*, vol. 37, no. web server issue, pp. W363–W368, Jul. 2009.
[37] 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 Mol. Biol.*, vol. 21, no. 3, pp. 305–318, Jun. 2012.

**Noah M. Daniels** received the PhD degree in computer science from Tufts University in 2013. He is currently a postdoctoral associate in the Department of Mathematics and the Computer Science and Artificial Intelligence Lab at MIT. His research interests include computational molecular biology, algorithms, and functional programming.

**Andrew Gallant** received the MS degree in computer science in 2012, where he is a doctoral candidate in the Department of Computer Science at Tufts University. His research interests include computational molecular biology and functional programming.

**Norman Ramsey** received the PhD degree in computer science at Princeton in 1993. He is an associate professor of computer science at Tufts, which he joined after eight years as an assistant and associate professor at Harvard. He has also held faculty appointments at the University of Virginia and at Purdue, as well as research positions at Bellcore, Bell Labs, and Microsoft Research. He was a Hertz fellow and an Alfred P. Sloan Research fellow. His work spans the range from theory to practice and covers primarily programming languages and software engineering. He is best known for work on low-level programming-language infrastructure: code generation, debugging, linking, binary translation, register allocation, and so on.

**Lenore J. Cowen** received the PhD degree in applied mathematics from MIT in 1993. She is currently a professor of computer science at Tufts University, with a joint appointment in the Tufts Mathematics Department. She has been a US National Science Foundation (NSF) postdoctoral fellow, an ONR Young investigator, and a fellow of the Radcliffe Institute for Advanced Study. Her research interests span three areas: discrete mathematics (since high school), algorithms (since graduate school), and computational molecular biology (since 2000).