EFFECTIVE OPTIMIZATION ALGORITHMS FOR FRAGMENT-ASSEMBLY BASED PROTEIN STRUCTURE PREDICTION

Kevin W. DeRonne* and George Karypis
Department of Computer Science & Engineering,
Digital Technology Center, Army HPC Research Center
University of Minnesota, Minneapolis, MN 55455
**Email: {deronne, karypis}@cs.umn.edu*

Despite recent developments in protein structure prediction, an accurate new fold prediction algorithm remains elusive. One of the challenges facing current techniques is the size and complexity of the space containing possible structures for a query sequence. Traditionally, to explore this space fragment assembly approaches to new fold prediction have used stochastic optimization techniques. Here we examine deterministic algorithms for optimizing scoring functions in protein structure prediction. Two previously unused techniques are applied to the problem, called the Greedy algorithm and the Hill-climbing algorithm. The main difference between the two is that the latter implements a technique to overcome local minima. Experiments on a diverse set of 276 proteins show that the Hill-climbing algorithms consistently outperform existing approaches based on Simulated Annealing optimization (a traditional stochastic technique) in optimizing the root mean squared deviation (RMSD) between native and working structures.

1. INTRODUCTION

Reliably predicting protein structure from amino acid sequence remains a challenge in bioinformatics. Although the number of known structures continues to grow, many new sequences still lack a known homolog in the PDB [2], which makes it harder to predict structures for these sequences. The conditional existence of a known structural homolog to a query sequence commonly delineates a set of subproblems within the greater arena of protein structure prediction. For example, the biennial CASP competition[a] breaks down structure prediction as follows. In homologous fold recognition the structure of the query sequence is similar to a known structure for some other sequence. However, these two sequences have only a low (though detectable) similarity. In analogous fold recognition there exists a known structure similar to the correct structure of the query, but the sequence of that structure has no detectable similarity to the query sequence. Still more challenging is the problem of predicting the structure of a query sequence lacking a known structural relative, which is called new fold (NF) prediction.

Within the context of the NF problem knowledge-based methods have attracted increasing attention over the last decade. In CASP, prediction approaches that assemble fragments of known structures into a candidate structure [18, 7, 10] have consistently outperformed alternative methods, such as those based largely on explicit modeling of physical forces. Fragment assembly for a query protein begins with the selection of structural fragments based on sequence information. These fragments are then successively inserted into the query protein's structure, replacing the coordinates of the query with those of the fragment. The quality of this new structure is assessed by a scoring function. If the scoring function is a reliable measure of how close the working structure is to the native fold of the protein, then optimizing the function through fragment insertions will produce a good structure prediction. Thus, building a structure in this manner can break down into three main components: a fragment selection technique, an optimizer for the scoring function, and the scoring function itself.

To optimize the scoring function, all the leading assembly-based approaches use an algorithm involving a stochastic search (e.g. Simulated Annealing [18], genetic algorithms [7], or conformational space annealing [10]). One potential drawback of such techniques is that they can require extensive parameter tuning before producing good solutions.

*Corresponding author.
[a]http://predictioncenter.org/

In this paper we wish to examine the relative performance of deterministic and stochastic techniques to optimize a scoring function. The new algorithms presented below are inspired by techniques originally developed in the context of graph partitioning [4], and do not depend on a random element. The Greedy approach examines all possible fragment insertions at a given point and chooses the best one available. The Hill-climbing algorithm follows a similar strategy but allows for moves that reduce the score locally, provided that they lead to a better global score.

Several variables can affect the performance of optimization algorithms in the context of fragment-based *ab initio* structure prediction. For example, how many fragments per position are available to the optimizer, how long the fragments are, if they should be multiple sizes at different stages [18] or all different sizes used together [7], and other parameters specific to the optimizer can all influence the quality of the resulting structures.

Taking the above into account, we varied fragment length and number of fragments per position when comparing the performance of our optimization algorithms to that of a tuned Simulated Annealing approach. Our experiments test these algorithms on a diverse set of 276 protein domains derived from SCOP 1.69 [14]. The results of these experiments show that the Hill-climbing-based approaches are very effective in producing high-quality structures in a moderate amount of time, and that they generally outperform Simulated Annealing. On the average, Hill-climbing is able to produce structures that are 6% to 20% better (as measured by the root mean square deviation (RMSD) between the computed and its actual structure), and the relative advantage of Hill-climbing-based approaches improves with the length of the proteins.

2. MATERIALS AND METHODS

2.1. Data

The performance of the optimization algorithms studied in this paper were evaluated using a set of proteins with known structure that was derived from SCOP 1.69 [14] as follows. Starting from the set of domains in SCOP, we first removed all membrane and cell surface proteins, and then used Astral's tools [3] to construct a set of proteins with less than 25% sequence identity. This set was further reduced by keeping only the structures that were determined by X-ray crystallography, filtering out any proteins with a resolution greater than 2.5Å, and removing any proteins with a $C_\alpha - C_\alpha$ distance greater than 3.8Å times their sequential separation[b].

The above steps resulted in a set of 2817 proteins. From this set, we selected a subset of 276 proteins (roughly 10%) to be used in evaluating the performance of the various optimization algorithms (i.e., a test set), whereas the remaining 2541 sequences were used as the database from whence to derive the structural fragments (i.e., a training set).[c] The test sequences, whose characteristics are summarized in Table 1, were selected to be diverse in length and secondary structure composition.

Table 1. Number of sequences at various length intervals and SCOP class.

	Sequence Length			
SCOP Class	< 100	100–200	> 200	total
alpha	23	40	6	69
beta	23	27	18	69
alpha/beta	4	26	39	69
alpha+beta	15	36	17	69

2.2. Neighbor Lists

As the search space for fragment assembly is much too vast, fragment-based *ab initio* structure prediction approaches must reduce the number of possible structures that they consider. They accomplish this primarily by restricting the number of structural fragments that can be used to replace each k-mer of the query sequence. In evaluating the various optimization algorithms developed in this work, we followed a methodology for identifying these structural fragments that is similar in spirit to that used by the Rosetta [18] system.

Consider a query sequence X of length l. For

[b]No bond lengths were modified to fit this constraint; proteins not satisfying it were simply removed from consideration.
[c]This dataset is available at http://www.cs.umn.edu/~deronne/supplement/optimize

each position i, we identify a list (L_i) of n structural fragments by comparing the query sequence against the sequences of the proteins in the training set. For fragments of length k, these comparisons involve the k-mer of X starting at position i ($0 \leq i \leq l-k+1$) and all k-mers in the training set. The n structural fragments are selected so that their corresponding sequences have the highest profile-based score with the query sequence's k-mer. Throughout the rest of this paper, we will refer to the list L_i as the *neighbor list* of position i.

In our study we used neighbor lists containing fragments of a single length as well as neighbor lists containing fragments of different lengths. In the latter case we consider two different approaches to leveraging the varied length fragments. The first, referred to as *scan*, uses the fragment lengths in decreasing order. For example, if the neighbor lists contain structural fragments of length three, six, and nine, the algorithm starts by first optimizing the structure using only fragments of length nine, then fragments of length six, and finally fragments of length three. Each one of these optimization phases terminates when the algorithm has finished (i.e., reached a local optimum or performed a predetermined number of iterations), and the resulting structure becomes the input to the subsequent optimization phase. The second approach for combining different length fragments is referred to as *pool*, and it optimizes the structure once, selecting fragments from any available length. Using any single length fragment in isolation, or using either scan or pool will be referred to as a *fragment selection scheme*.

2.2.1. *Sequence Profiles*

The comparisons between the query and the training sequences take advantage of evolutionary information by utilizing PSI-BLAST [1] generated sequence profiles.

The profile of a sequence X of length l is represented by two $l \times 20$ matrices. The first is its position-specific scoring matrix PSSM_X that is computed directly by PSI-BLAST. The rows of this matrix correspond to the various positions in X, while the columns correspond to the 20 distinct amino acids. The second matrix is its position-specific *frequency* matrix PSFM_X that contains the frequencies used by PSI-BLAST to derive PSSM_X. These frequencies (also referred to as *target frequencies* [13]) contain both the sequence-weighted observed frequencies (also referred to as *effective frequencies* [13]) and the BLOSUM62 [6] derived-pseudocounts [1]. For each row of a PSFM, the frequencies are scaled so that they add up to one. In the cases where PSI-BLAST could not produce meaningful alignments for a given position of X, the corresponding rows of the two matrices are derived from the scores and frequencies of BLOSUM62.

For our study, we used the version of the PSI-BLAST algorithm available in NCBI's blast release 2.2.10 to generate profiles for both the test and training sequences. These profiles were derived from the multiple sequence alignment constructed after five iterations using an e value of 10^{-2}. The PSI-BLAST search was performed against NCBI's nr database that was downloaded in November of 2004 and which contained 2,171,938 sequences.

2.2.2. *Profile-to-Profile Scoring Method*

The similarity score between a pair of k-mers (one from the query sequence and one from a sequence in the training set) was computed as the ungapped alignment score of the two k-mers whose aligned positions were scored using profile information.

Many different schemes have been developed for determining the similarity between profiles that combine information from the original sequence, position-specific scoring matrix, or position-specific target and/or effective frequencies [13, 21, 11]. In our work we use a scheme that is derived from PICASSO [5, 13] that was recently used in developing effective remote homology prediction and fold recognition algorithms [16]. Specifically, the similarity score between the ith position of protein X's profile, and the jth position of protein Y's profile is given by

$$S_{X,Y}(i,j) = \sum_{l=1}^{20} \mathrm{PSFM}_X(i,l)\,\mathrm{PSSM}_Y(j,l) \; + \; \sum_{l=1}^{20} \mathrm{PSFM}_Y(j,l)\,\mathrm{PSSM}_X(i,l), \tag{1}$$

where $\mathrm{PSFM}_X(i,l)$ and $\mathrm{PSSM}_X(i,l)$ are the values corresponding to the lth amino acid at the ith position of X's position-specific scoring and frequency

matrices. $\text{PSFM}_Y(j,l)$ and $\text{PSSM}_Y(j,l)$ are defined in a similar fashion.

Equation 1 determines the similarity between two profile positions by weighting the position-specific scores of the first sequence according to the frequency at which the corresponding amino acid occurs in the second sequence's profile. The key difference between Equation 1 and the corresponding scheme used in [13] (therein referred to as PICASSO3), is that our measure uses the target frequencies, whereas the scheme of [13] is based on effective frequencies.

2.3. Protein Structure Representation

Internally, we consider only the positions of the C_α atoms, and we use a vector representation of the protein in lieu of ϕ and ψ backbone angles. Our protein construction approach uses the actual coordinates of the atoms in each fragment, rotated and translated into the reference frame of the working structure. Fragments are taken directly from known structures, and are chosen from the training dataset using the above profile-profile scoring methods.

2.4. Scoring Function

As the focus of this work is to develop and evaluate new optimization techniques, we use the RMSD between the predicted and native structure of a protein as the scoring function. Although such a function cannot serve as a predictive measure, we believe that using this as a scoring function allows for a clearer differentiation between the optimization process and the scoring function. In effect, we assume an ideal scoring function in order to test the optimization techniques.

2.5. Optimization Algorithms

In this study we compare the performance of three different optimization algorithms in the context of fragment assembly-based approaches for *ab initio* structure predictions. One of these algorithms, Simulated Annealing [8], is currently a widely used method to solve such problems, whereas the other two algorithms, Greedy and Hill-climbing, are newly developed for this work.

The key operation in all three of these algorithms is the replacement of a k-mer starting at a particular position i, with that of a neighbor structure. We will refer to this operation as a *move*. A move is considered valid if, after inserting the fragment, it does not create any steric conflicts. A structure is considered to have a steric conflict if it contains a pair of C_α atoms within 2.5Å of one another. Also, for each valid move, its *gain* is defined as the improvement in the value of the scoring function between the working structure and the native structure of the protein.

2.5.1. *Simulated Annealing (SA)*

Simulated Annealing [8] is a generalization of the Monte Carlo [12] method for discrete optimization problems. This optimization approach is designed to mimic the process by which a material such as metal or glass cools. At high temperatures, the atoms of a metal can adopt configurations not available to them at lower temperatures—e.g., a metal can be a liquid rather than a solid. As the system cools, the atoms arrange themselves into more stable states, forming a stronger substance.

The Simulated Annealing (SA) algorithm proceeds in a series of discrete steps. In each step it randomly selects a valid move and performs it (i.e., inserts the selected fragment into the structure). This move can either improve or degrade the quality of the structure. If the move improves the quality, then the move is accepted. If it degrades the quality, then the move will still be accepted with probability

$$p = e^{\left(\frac{S_{old}-S_{new}}{T}\right)}, \qquad (2)$$

where T is the current temperature of the system, q_{old} is the score of the last state, and q_{new} is the score of the state in question. From Equation 2 we see that the likelihood of accepting a bad move is inversely related to the temperature and how much worse the new structure is from the current structure. That is, the optimizer will accept a very bad move with a higher probability if the temperature is high than if the temperature is low.

The algorithm begins with a high system temperature which it progressively decreases according to an *annealing schedule*. As the optimization must use finite steps, the cooling of the system cannot be

continuous, but the annealing schedule can be modified to increase its smoothness. The annealing schedule depends on a combination of the number of total allowed moves and the number of steps in which to make those moves. Our implementation of Simulated Annealing, following the general framework employed in Rosetta [18], uses an annealing schedule that linearly decreases the temperature of the system to zero over a fixed number of cycles.

Simulated Annealing is a highly tunable optimization framework. The starting temperature and the annealing schedule can be varied to improve performance, and the performance of the algorithm depends greatly on these parameters. Section 3.2.1 describes how we arrive at the values for these parameters of SA as implemented in this study.

2.5.2. *The Greedy Algorithm (G)*

One of the characteristics of the Simulated Annealing algorithm is that it considers moves for insertion at random, irrespective of their gains. The Greedy algorithm that we present here selects maximum gain moves.

Specifically, the algorithm consists of two phases. In the first phase, called *initial structure generation*, the algorithm starts from a structure corresponding to a fully extended chain, and attempts to make a valid move at each position of the protein. This is achieved by scoring all neighbors in each neighbor list and inserting the best neighbor (i.e. the neighbor with the highest gain) from each list. If some positions have no valid moves on the first pass, the algorithm attempts to make moves at these positions after trying all positions once. This ensures that the algorithm makes moves at nearly every position down a chain, and also provides a good starting point for the next phase.

In the second phase, called *progressive refinement*, the algorithm repeatedly finds the maximum gain valid move over all positions of the chain, and if this move leads to a positive gain—i.e. it improves the value of the scoring function—the algorithm makes the move. This progressive refinement phase terminates upon failing to find any move to make. The Greedy algorithm is guaranteed to finish the progressive refinement phase in at least a local optimum.

2.5.3. *Hill-Climbing (HC)*

The Hill-climbing algorithm was developed to allow the Greedy algorithm to effectively climb out of locally optimal solutions. The key idea behind Hill-climbing is to not stop after achieving a local optimum but to continue performing valid moves in the hope of finding a better local or a (hopefully) global optimum.

Specifically, the Hill-climbing algorithm works as follows. The algorithm begins by applying the Greedy algorithm in order to reach a local optimum. At this point, it begins a sequence of iterations consisting of a *hill-climbing* phase, followed by a progressive refinement phase (as in the Greedy approach). In the hill-climbing phase, the algorithm performs a series of moves, each time selecting the highest gain valid move irrespective of whether or not it leads to a positive gain. If at any point during this series of moves, the working structure achieves a score that is better than that of the structure at the beginning of the hill-climbing phase, this phase terminates and the algorithm enters the progressive refinement phase. The above sequence of iterations terminates when the hill-climbing phase is unable to produce a better structure after successively performing all best scoring valid moves.

Since the hill-climbing phase starts at a local optimum, its initial set of moves will lead to a structure whose quality (as measured by the scoring function) is worse than that at the beginning of the hill-climbing phase. However, subsequent moves can potentially lead to improvements that outweigh the initial quality degradation; thus allowing the algorithm to climb out of locally optimal solutions.

Move Locking As Hill-climbing allows negative gain moves, the algorithm can potentially oscillate between a local optimum and a non-optimal solution. To prevent this from happening, we implement a notion of move locking. After each move, a *lock* is placed on the move to prevent the algorithm from making this move again within the same phase. By doing so, we ensure the algorithm does not repeatedly perform the same sequence of moves; thus guaranteeing its termination after a finite number of moves. All locks are cleared at the end of a hill-

climbing phase, allowing the search maximum freedom to proceed.

We investigate two different locking methods. The first, referred to as *fine-grain locking*, locks the single move made. The algorithm can subsequently select a different neighbor for insertion at this position. The second, referred to as *coarse-grain locking*, locks the position of the query sequence itself; preventing any further insertions at that position. In the case of pooling, coarse locking locks moves of all sizes.

Since fine-grain locking is less restrictive, we expect it to lead to better quality solutions. However, the advantage of coarse-grain locking is that each successive fragment insertion significantly reduces the set of fragments that need to be considered for future insertions; thus, leading to a faster optimization algorithm.

2.5.4. *Efficient Checking of Steric Conflicts*

One characteristic of the Greedy and Hill-climbing algorithms is their need to evaluate the validity of every available move after every insertion. This proves necessary because each insertion can potentially introduce new proximity conflicts. In an attempt to assuage the time requirement for this process, we have developed an efficient formulation for validity checking.

Recall that a valid move brings no two C_α atoms within 2.5Å of each other. To quickly determine if this proximity constraint holds, we impose a three-dimensional grid over the structure being built with boxes 2.5Å on each side. As each move is made, its atoms are added to the grid, and for each addition the surrounding 26 boxes are checked for atoms violating the proximity constraint. In this fashion we limit the number of actual distances that must be computed.

We further decrease the required time by sequentially checking neighbors at each position down the amino acid chain. All atoms upstream of the insertion point must be internally valid, as they have previously passed proximity checks. Thus, we need only examine those atoms at or downstream from the insertion. This saves on computation time within one iteration of checking all possible moves.

3. EXPERIMENTAL EVALUATION

3.1. Performance of the Greedy and Hill-climbing Algorithms

To compare the effectiveness of the Greedy and Hill-climbing optimization techniques, we report results from a series of experiments in which we vary a number of parameters. Table 2 shows results for the Greedy and Hill-climbing optimization techniques using k-mer sizes of 9, 6, and 3 individually, as well as using the scan and pool techniques to combine them. Average times are also reported for each of these five fragment selection schemes.

Examining Table 2, we see that the Hill-climbing algorithm consistently outperforms the Greedy algorithm. As Hill-climbing includes running Greedy to convergence, the result is not surprising, and neither is the increased run-time that Hill-climbing requires. Both schemes seem to take advantage of the increased flexibility of smaller fragments and greater numbers of fragments per position. For example, on the average the 3-mer results are 9.4%, 12.0%, and 8.5% better than the corresponding 9-mer results for Greedy, Hill-climbing (coarse) (hereafter HC_c) and Hill-climbing (fine) (hereafter HC_f), respectively. Similarly, increasing the neighbor lists from 25 to 100 yields a 23.1%, 31.6%, and 43.6% improvement for Greedy, HC_c, and HC_f, respectively. These results also show that the search algorithms embedded in Greedy, HC_c, and HC_f are progressively more powerful as the size of the overall search space increases.

With respect to locking, a less restrictive fine-grained approach generally yields better results than a coarse-grained scheme. For example, averaging over all experiments, fine-grained locking yields a 21.2% improvement over coarse-grained locking. However, this increased performance comes at the cost of an increase in run-time of 1128% on the average.

Comparing the performance of the scan and pooling methods to combine variable length k-mers we see that pool performs consistently better than scan by an average of 4.4%. This improvement also comes at the cost of an increase in run time, which in this case is 131.1% on the average. Results from the pool and scan settings clearly indicate that Greedy

Table 2. Average values over 276 proteins optimized using Hill-climbing and different locking schemes. Times are in seconds and scores are in Å. Lower is better in both cases.

		$n = 25$		$n = 50$		$n = 75$		$n = 100$	
		Score	Time	Score	Time	Score	Time	Score	Time
Greedy	$k = 9$	9.07	11	8.20	14	7.77	18	7.38	22
	$k = 6$	8.76	12	7.98	17	7.50	22	7.21	27
	$k = 3$	8.20	15	7.51	22	7.08	30	6.80	39
	Scan	7.21	33	6.52	40	6.02	48	5.81	56
	Pool	7.06	41	6.34	58	5.94	76	5.57	97
Hill-climbing (coarse) (HC_c)	$k = 9$	6.70	49	5.99	98	5.54	143	5.29	226
	$k = 6$	6.46	65	5.67	124	5.23	221	4.93	279
	$k = 3$	6.07	76	5.35	182	4.92	313	4.68	433
	Scan	5.10	120	4.50	216	4.01	333	3.76	517
	Pool	5.06	341	4.33	912	3.96	1588	3.74	1833
Hill-climbing (fine) (HC_f)	$k = 9$	5.81	357	4.96	1314	4.53	2656	4.30	4978
	$k = 6$	5.67	352	4.76	1417	4.30	3277	3.99	5392
	$k = 3$	5.56	390	4.60	1561	4.10	3837	3.87	6369
	Scan	4.69	748	3.92	2878	3.37	6237	3.17	10677
	Pool	4.30	1997	3.56	7101	3.14	18000	2.87	21746

and HC_c are not as effective at exploring the search space as HC_f.

Table 3. SCOP classes and lengths for the tuning set.

SCOP identifier	length	SCOP class
d1jiwi_	105	beta
d1kpf__	111	alpha+beta
d2mcm__	112	beta
d1bea__	116	alpha
d1ca1_2	121	beta
d1jiga_	146	alpha
d1nbca_	155	beta
d1yaca_	204	alpha/beta
d1a8d_2	205	beta
d1aoza2	209	beta

3.2. Comparison with Simulated Annealing

3.2.1. *Tuning the Performance of SA*

Due to the sensitivity of Simulated Annealing to specific values for various parameters, we performed a search on a subset of the test proteins in an attempt to maximize the ability of SA to optimize the test structures. Specifically, we attempted to find values for two governing factors: the initial temperature T_0 and the number of moves nm. To this end, we selected ten medium length proteins of diverse secondary structural classification (see Table 3), and optimized them over various initial temperatures. The initial temperature that yielded the best average optimized RMSD was $T_0 = 0.1$ and we used this value in all subsequent experiments.

In addition to an initial temperature, when using Simulated Annealing one must select an appropriate annealing schedule. Our annealing schedule decreases the temperature linearly over 3500 cycles. This allows for a smooth cooling of the system. Over the course of these cycles, the algorithm attempts $\alpha \times (l \times n)$ moves, where α is an empirically determined scaling factor, l is the number of amino acids in the query protein, and n is the number of neighbors per position. Note that for the scan and pool techniques (see Section 2.2), we allow SA three times the number of attempted moves because the total number of neighbors is that much larger. In order to produce comparable run-times to the G, HC_c and HC_f schemes, α values of 20, 50 and 100 are employed, respectively. Finally, following recent work [17] we allowed for a temporary increase in the temperature after 150 consecutive rejected moves.

3.2.2. *Results*

The Simulated Annealing results are summarized in Table 4. As we see in this table, Simulated Annealing consistently outperforms the Greedy scheme. Specifically, the average performance of SA with $\alpha = 20$ is 15.1% better than that obtained by G. These performance comparisons are obtained by averaging the

Table 4. Average values over 276 proteins optimized using Simulated Annealing. Times are in seconds and scores are in Å. Lower is better in both cases.

		$n = 25$		$n = 50$		$n = 75$		$n = 100$	
		Score	Time	Score	Time	Score	Time	Score	Time
$\alpha = 20$	$k = 9$	7.88	25	6.99	31	6.54	36	6.28	42
	$k = 6$	7.45	25	6.46	30	6.12	36	6.03	42
	$k = 3$	6.78	25	6.01	31	5.87	37	5.81	43
	Scan	6.11	74	5.54	92	5.39	109	5.39	128
	Pool	5.93	75	5.84	94	6.00	112	6.13	132
$\alpha = 50$	$k = 9$	7.20	34	6.44	48	6.31	65	6.21	80
	$k = 6$	6.69	34	6.13	49	6.06	64	6.11	80
	$k = 3$	6.19	35	5.90	51	6.02	67	6.18	81
	Scan	5.68	103	5.48	148	5.50	197	5.48	258
	Pool	5.91	103	6.08	150	6.25	203	6.31	251
$\alpha = 100$	$k = 9$	6.76	52	6.34	81	6.31	112	6.28	145
	$k = 6$	6.31	50	6.14	81	6.18	115	6.26	146
	$k = 3$	6.05	52	6.21	84	6.34	118	6.40	155
	Scan	5.65	148	5.53	241	5.62	348	5.62	439
	Pool	5.99	156	6.23	265	6.34	352	6.38	447

The values of α in the above table scale the number of moves Simulated Annealing is allowed to make. In our case, the total number of moves is $\alpha \times (l \times n)$ where l is the length of the protein being optimized and n is the number of neighbors per position.

Table 5. Average values over the longest 138 proteins optimized using Hill-climbing and different locking schemes. Times are in seconds and scores are in Å. Lower is better in both cases.

		$n = 25$		$n = 50$		$n = 75$		$n = 100$	
		Score	Time	Score	Time	Score	Time	Score	Time
Greedy	$k = 9$	11.56	17	10.59	23	10.01	29	9.52	37
	$k = 6$	11.15	19	10.29	27	9.77	36	9.52	46
	$k = 3$	10.36	24	9.73	38	9.30	51	8.95	68
	Scan	9.52	50	8.62	62	8.08	76	7.78	91
	Pool	9.24	64	8.50	95	7.96	126	7.55	164
Hill-climbing (coarse) (HC_c)	$k = 9$	8.44	90	7.48	185	6.89	271	6.46	433
	$k = 6$	8.11	121	7.16	234	6.63	424	6.18	535
	$k = 3$	7.60	142	6.86	347	6.32	602	6.07	833
	Scan	6.43	213	5.73	394	5.08	625	4.72	982
	Pool	6.42	651	5.55	1773	4.93	3109	4.74	3581
Hill-climbing (fine) (HC_f)	$k = 9$	7.33	672	6.18	2477	5.55	4992	5.23	9396
	$k = 6$	7.23	662	5.92	2690	5.31	6238	4.95	10252
	$k = 3$	7.02	737	5.88	2974	5.24	7360	4.94	12190
	Scan	6.03	1376	4.97	5173	4.18	11524	3.94	19818
	Pool	5.38	3844	4.45	13717	3.82	34960	3.40	42045

ratios between the two schemes of the corresponding RMSDs over all fragment selection schemes and values of n. The superior performance of Simulated Annealing over Greedy is to be expected, as Greedy lacks any sort of hill-climbing ability, whereas the stochastic nature of Simulated Annealing allows it a chance of overcoming locally optimal solutions. In contrast, both the fine and coarse-locking versions of Hill-climbing outperform SA. More concretely, on the average HC_c performs 22.0% better than SA with $\alpha = 50$, and HC_f performs 46.3% better than SA with $\alpha = 100$.

Analyzing the performance of Simulated Annealing with respect to the value of α, we see that while Simulated Annealing shows an average improvement of 1.7% when α is increased from 20 to 50, the performance deteriorates by an average of 0.07% when α is increased from 50 to 100. This indicates that further increasing the value of α may not lead to performance comparable to that of the Greedy and Hill-climbing schemes.

Also note that in some of the results shown in

Table 6. Average values over the longerst 138 proteins optimized using Simulated Annealing. Times are in seconds and scores are in Å. Lower is better in both cases.

		$n = 25$		$n = 50$		$n = 75$		$n = 100$	
		Score	Time	Score	Time	Score	Time	Score	Time
$\alpha = 20$	$k = 9$	9.97	37	8.92	48	8.38	58	8.16	68
	$k = 6$	9.54	37	8.39	48	7.94	58	7.89	69
	$k = 3$	8.63	38	7.77	49	7.70	59	7.67	70
	Scan	7.86	113	7.20	145	7.06	176	7.12	210
	Pool	7.76	114	7.80	147	8.07	180	8.18	215
$\alpha = 50$	$k = 9$	9.21	53	8.30	80	8.26	109	8.23	135
	$k = 6$	8.58	54	8.01	81	8.06	108	8.16	136
	$k = 3$	7.96	55	7.76	83	7.95	112	8.16	138
	Scan	7.31	164	7.18	245	7.25	331	7.24	440
	Pool	7.90	163	8.19	248	8.33	341	8.39	427
$\alpha = 100$	$k = 9$	8.57	86	8.35	137	8.41	192	8.43	251
	$k = 6$	8.17	83	8.19	138	8.23	197	8.36	254
	$k = 3$	7.90	86	8.20	143	8.30	204	8.37	270
	Scan	7.32	243	7.33	411	7.39	592	7.45	760
	Pool	8.11	260	8.30	455	8.36	606	8.44	778

The values of α in the above table scale the number of moves Simulated Annealing is allowed to make. In our case, the total number of moves is $\alpha \times (l \times n)$ where l is the length of the protein being optimized and n is the number of neighbors per position.

Table 4, the performance occasionally decreases as the α value increases. This ostensibly strange result comes from the dependence of the cooling process on the number of allowed moves, in which the value of α plays a role. For all entries in Table 4 the annealing schedule will cool the system over a fixed number of steps, but the number of moves made will vary greatly. Thus, in order to keep the cooling of the system linear we vary the number of moves allowed before the system reduces its temperature. As a result, different values of α can lead to different randomly chosen optimization paths.

Comparing the performance of the various optimization schemes with respect to the various fragment selection schemes, we see an interesting trend. The performance of SA deteriorates (by 9.6% on the average) when the different length k-mers are used via the pool method, whereas the performance of HC_f improves (by 4.4% on average). We are currently investigating the source of this behavior, but one possible explanation is that Simulated Annealing has a bias towards smaller fragments. This bias might result because an insertion of a bad 3-mer will degrade the structure less than that of a bad 9-mer, and as a result, the likelihood of accepting the former move will be higher (Equation 2). This may reduce the optimizers ability to effectively utilize the variable length k-mers.

Performance on Longest Sequences In order to gain a better understanding of how the optimization schemes perform, we focus on the longer half of the test proteins. Average RMSDs and times for the Greedy and Hill-climbing schemes are shown in Table 5, and average RMSDs and times for Simulated Annealing are shown in Table 6.

In general, the trends in these tables agree with the trends in the average values over all the proteins. However, one key difference is that the relative improvement of the Hill-climbing scheme over Simulated Annealing is higher, while Greedy actually does worse. For example, comparing G and SA for $\alpha = 20$, SA performs 15.7% better, as opposed to 15.1% for the full average. Comparing with SA for $\alpha = 50$, HC_c performs 27.0% better as opposed to 22.0% for the full average. Finally, comparing with SA for $\alpha = 100$, HC_f is 54.6% better, as opposed to 46.3% for the full average. These results suggest that, in the context of a larger search space, a hill-climbing ability is important, and that the hill-climbing abilities of HC_c and HC_f are better than those of SA.

4. DISCUSSION AND CONCLUSIONS

This paper presents two new techniques for optimizing scoring functions for protein structure predic-

tion. One of these approaches, HC_c, using the scan technique, reaches better solutions than Simulated Annealing in comparable time. The performance of SA seems to saturate beyond $\alpha = 50$, but HC_f will make use of an increased time allowance, finding the best solutions of all the examined algorithms. Furthermore, experiments with variations on the number of moves available to the optimizer demonstrate that the Hill-climbing approach makes better use of an expanded search space than Simulated Annealing. Additionally, Simulated Annealing requires the hand-tuning of several parameters, including the total number of moves, the initial temperature, and the annealing schedule. One of the main advantages of using schemes like Greedy and Hill-climbing is that they do not rely on such parameters.

Recently, greedy techniques have been applied to problems similar to the one this paper addresses. The first problem is to determine a set of representative fragments for use in decoy structure construction [15, 9]. The second problem is to reconstruct a native protein fold given such a set of representative fragments [19, 20]. The greedy approaches used for both these problems traverse the query sequence in order, inserting the best found fragment for each position. As an extension, the algorithms build multiple structures simultaneously in the search for a better structure. While such approaches have the ability to avoid local minima, they lack an explicit notion of hill-climbing.

The techniques this paper describes could be modified to solve either of the above two problems. To build a representative set of fragments, one could track the frequency of fragment use within multiple Hill-climbing optimizations of different proteins. This would yield a large set of fragments, which could serve as input to a clustering algorithm. The centroids of these clusters could then be used in decoy construction. In order to construct a native fold from these fragments one need only restrict the move options of Hill-climbing to the representative set. We are currently working on adapting our algorithms to solve these problems.

ACKNOWLEDGMENTS

This work was supported in part by NSF EIA-9986042, ACI-0133464, IIS-0431135, and NIH RLM008713A; the Digital Technology Center at the University of Minnesota; and by the Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under Cooperative Agreement number DAAD19-01-2-0014. The content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Access to research and computing facilities was provided by the Digital Technology Center and the Minnesota Supercomputing Institute.

References

1. S. F. Altschul, L. T. Madden, A. A. Schffer, 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):3389–402, 1997.
2. H.M. Berman, J.Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Research*, 2000.
3. J. Chandonia, G. Hon, N. S. Walker, L. Lo Conte, P. Koehl, M. Levitt, and S. E. Brenner. The astral compendium in 2004. *Nucleic Acids Research*, 2004.
4. C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. *Proceedings of the 19th Design Automation Conference*, 1982.
5. A. Heger and L. Holm. Picasso: generating a covering set of protein family profiles. *Bioinformatics*, 2001.
6. S. Henikoff and J. G. Henikoff. Amino acid subsitution matrices from protein blocks. *PNAS*, 89:10915–10919, 1992.
7. K. Karplus, R. Karchin, J. Draper, J. Casper, Y. Mandel-Gutfreund, M. Diekhans, and R. Hughey. Combining local-structure, fold-recognition, and new fold methods for protein structure prediction. *PROTEINS: Structure, Function and Genetics*, 2003.
8. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
9. R. Kolodny, P. Koehl, L. Guibas, and M. Levitt. Small libraries of protein fragments model native protein structures accurately. *Journal of Molecular Biology*, 323:297–307, 2002.
10. J. Lee, S. Kim, K. Joo, I. Kim, and J. Lee. Prediction of protein tertiary structure using profesy, a novel method based on fragment assembly and conformational space annealing. *PROTEINS: Structure, function and bioinformatics*, 2004.

11. M. Marti-Renom, M. Madhusudhan, and A. Sali. Alignment of protein sequences by their profiles. *Protein Science*, 13:1071–1087, 2004.
12. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.
13. D. Mittelman, R. Sadreyev, and N. Grishin. Probabilistic scoring measures for profile-profile comparison yield more accurate short seed alignments. *Bioinformatics*, 19(12):1531–1539, 2003.
14. A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.
15. B. H. Park and M. Levitt. The complexity and accuracy of discrete state models of protein structure. *Journal of Molecular Biology*, 249:493–507, 1995.
16. H. Rangwala and G. Karypis. Profile based direct kernels for remote homology detection and fold recognition. *Bioinformatics*, 21:4239–4247, 2005.
17. C. A. Rohl, C. E. M. Strauss, K. M. S. Misura, and D. Baker. Protein structure prediction using rosetta. *Methods in Enzymology*, 2004.
18. K. T. Simons, C. Kooperberg, E. Huang, and D. Baker. Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and bayesian scoring functions. *Journal of Molecular Biology*, 1997.
19. P. Tuffery and P. Derreumaux. Dependency between consecutive local conformations helps assemble protein structures from secondary structures using go potential and greedy algorithm. *Protein Science*, 61:732–740, 2005.
20. P. Tuffery, F. Guyon, and P. Derreumaux. Improved greedy algorithm for protein structure reconstruction. *Journal of Computational Chemistry*, 26:506–513, 2005.
21. G. Wang and R. L. Dunbrack JR. Scoring profile-to-profile sequence alignments. *Protein Science*, 13:1612–1626, 2004.