Genetic Algorithms For Protein Threading

Jacqueline Yadgari[#], Amihood Amir[#], Ron Unger*

[#]Department of Mathematics and Computer Science
*Department of Life Sciences
Bar-Ilan University
Ramat-Gan, 52900, Israel
*ron@biocom1.ls.biu.ac.il

Abstract

Despite many years of efforts, a direct prediction of protein structure from sequence is still not possible. As a result, in the last few years researchers have started to address the "inverse folding problem": Identifying and aligning a sequence to the fold with which it is most compatible, a process known as "threading". In two meetings in which protein folding predictions were objectively evaluated, it became clear that threading as a concept promises a real breakthrough, but that much improvement is still needed in the technique itself. Threading is a NP-hard problem, and thus no general polynomial solution can be expected. Still a practical approach with demonstrated ability to find optimal solutions in many cases, and acceptable solutions in other cases, is needed. We applied the technique of Genetic Algorithms in order to significantly improve the ability of threading algorithms to find the optimal alignment of a sequence to a structure, i.e. the alignment with the minimum free energy. A major progress reported here is the design of a representation of the threading alignment as a string of fixed length. With this representation validation of alignments and genetic operators are effectively implemented. Appropriate data structure and parameters have been selected. It is shown that Genetic Algorithm threading is effective and is able to find the optimal alignment in a few test cases. Furthermore, the described algorithm is shown to perform well even without pre-definition of core elements. Existing threading methods are dependent on such constraints to make their calculations feasible. But the concept of core elements is inherently arbitrary and should be avoided if possible. While a rigorous proof is hard to submit yet an, we present indications that indeed Genetic Algorithm threading is capable of finding consistently good solutions of full alignments in search spaces of size up to 10^{70}.

Introduction

The protein folding problem is to correctly compute the three-dimensional structure of a protein from its amino acid sequence. The structure of proteins is the key to gain insight into their function. Since proteins are involved in all biological processes, an accurate prediction of protein structure will change dramatically modern biology, biotechnology and medicine .

The "alphabet" from which proteins are made is a set of 20 amino acids, every protein being a sequence of amino acids with typical size range between 100 and 300, drawn from this alphabet. The chemical and physical properties of a protein molecule depend on its three-dimensional structure which depends inherently on the amino acid sequence. It is widely assumed that proteins fold up in a way that minimizes their free energy, so the chain ends up with the "most comfortable" configuration available to it.

Today the structure of proteins is discovered by using X-Ray Crystallography and NMR spectroscopy. Up to now the three-dimensional structures of only about 5000 proteins have been solved, compared with about 90,000 known sequences.

A direct calculation of protein structure from its sequence is not possible since it requires minimization of a function of thousands of variables, with constants that have not accurately determined. Threading represents another approach to predicting the three dimensional fold of a protein sequence by recognizing a known structure with which the sequence might be compatible. A given sequence is threaded through a given target structure by searching for a sequence-structure alignment that places sequence residues into energetically preferred structural positions. Since amino acids that are located far away in the sequence might interact closely in the three dimensional space imposed by the structure, the standard sequence matching algorithms are not applicable here. Currently, approximations based on dynamic programming variants or Monte Carlo search algorithms are used to find the best sequence-structure alignment. Here we propose the use of Genetic Algorithms to search for sequence-structure alignment. We present a representation of the problem and a data structure to support it, and we show evidence that the procedure is effective. The focus of this report is to demonstrate the feasibility of the design, further work is underway to prove its effectiveness in an actual prediction setting.

Methods

Threading

Threading is one of the recognition methods for protein structure prediction (Inverse Protein Folding)[Lemer et al., 1995]. This algorithm "threads" the sequence of one protein through the known structure of another, looking for a reliable alignment with the lowest free energy value. Thus, given a library of folds, one can decide by comparing (after proper statistical normalization) these best alignments, which fold is most likely to be adopted by the sequence. Based on the observation that the number of folds exist in Nature seems to be limited, about 1000 different families [Chotia, 1992], it is quite likely that the unknown structure will be similar to a known structure. This method has proved itself in recognizing similarity of a sequence to a protein of known structure in the absence of detectable sequence similarity [e.g. Bryant, 1996, Lemer et al., 1995].

In designing a threading procedure one needs an algorithm to align the residues of the sequence with a structure, and a score function to evaluate the quality of the alignment. This study suggests a novel algorithm for alignments, we use here a standard score function for the evaluation .

Many threading algorithms divide each structure into "core" and "non-core" regions. This classification is based usually on considering the secondary structure elements of alpha helices and beta sheets as cores and the loop regions connecting them as non-cores. No insertions and deletions are allowed in the core regions, and the score function is usually calculated only based on residues assigned to core elements. This make the calculation much simpler, no gap penalties are needed to be included, and since the total size of core elements is significantly smaller than the size of the full structure, the search space becomes considerably smaller. While it is true that secondary structure elements are more conserved than loop regions, it is clear that loop regions also carry structural signal, and thus threading methods that can handle full alignments might have an important advantage. Note that in full threading, gap penalties for creating insertions/deletions must be part of the score function, and as a first approximation residues that are inserted or deleted are not considered in the energy calculation.

Current threading algorithms are based on: enumeration in very small systems [Bryant and Lawrence, 1993], frozen dynamic programming approximations [Taylor and Orengo, 1989], Monte Carlo variants like Gibbs sampling [Bryant and Altschul, 1995], Branch and bound search [Lathrop and Smith, 1996]. Threading is a statistical method, and thus a good prediction is based on detecting many favorable interactions. Even in the cases where the correct fold is predicted, the actual alignments, which are crucial to gain biological insight, are often incorrect [Lemer at. al., 1995]. Thus, we think that better algorithms are needed to find the best threading.

Problem size. There are many ways for threading one sequence on a given structure. Consider the following parameters:

Sequence length = k ; Structure length = n

Since the problem is similar to the combinatorial problem of finding the number of combinations to distribute k identical objects into n different cells, the number of possible combinations is :

$$\frac{(n-1+k)!}{(n-1)!\,k!} \quad \text{(eq. 1.)}$$

This number is actually the size of our search space. By using core elements, i.e. restricting the possibilities for insertions and deletions only to loop regions, the search space size is reduced significantly.

Sequence length = k

Structure length = n

Total core length = c

Total loop length = $k - c$

Total number of core elements = l

Total number of loop elements = $l + 1 = m$

The number of possible combinations will be:

$$\frac{(m-1+(k-c))!}{(m-1)!(k-c)!} \quad \text{(eq. 2.)}$$

The sequence-structure threading problem is NP-complete. Protein folding is known to be a hard problem and formal models of the problems were shown to be NP-complete [e.g. Unger and Moult, 1993]. There was a hope that by inverting the folding problem and formulating it as a threading problem, its formal complexity will be lower. Unfortunately, this is not the case.

The search for the optimal alignment of a sequence to a known structure is proved to be hard [Lathrop, 1994], even in the case where core elements are used and the degrees of freedom are as described above (eq. 2.). The decision protein threading problem is NP-complete and the corresponding problem of finding the globally optimal protein threading is NP-hard.

Score function using protein core segments. Knowledge based potentials and energy functions are extracted from a database of known protein structures. These are based on the analysis of known three-dimensional structures of proteins using statistical procedures whose roots are in statistical physics [Sippl, 1995]. Most energy functions for threading are based on counts of frequency of contacts in the database of known structures. The assumption is that if a contact is frequent, it is energetically favorable.

In this work we used residue contact potential [Bryant and Lawerence, 1993] which gives contact energies as a function of residue type and distance interval, plus a

singleton potential that reflects the tendency of each residue to be buried below the surface of the molecule (hydrophobicity). In this potential only close contacts in three-dimension, that are non-local in the sequence, are considered.

This energy function is based on core segment elements, see fig. 1.

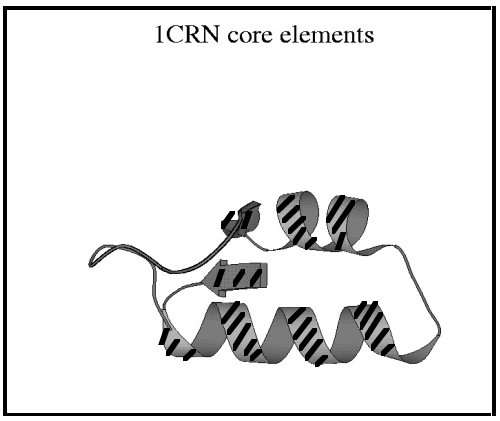
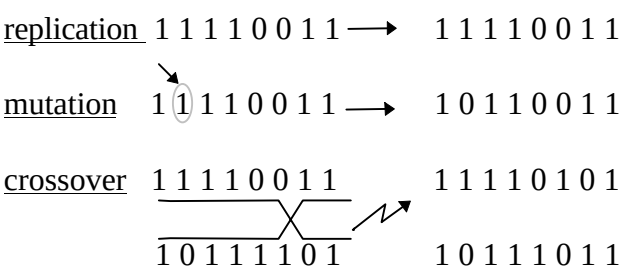


Fig. 1 Crambin core segments

Protein atomic coordinates were taken from the Protein Data Bank (PDB) [Bernstein et al., 1977].

Genetic Algorithms

Genetic Algorithms are a parallel computational approach based on the observation that natural processes adapt optimally to their environment in response to natural selection. Solutions are represented as strings. These strings are maintained as a population and allowed to interact. The interactions are via "genetic operators" such as: Replication, Mutation, and Crossover.



An evaluation function attaches "fitness" value to a solution as an indicator to its performance. Strings which represent solutions with higher values have a higher chance to participate in genetic operations.

Performing this process for many generations leads, in most cases, to the emergence of solutions with very high fitness [Goldberg, 1989, Holland, 1975]. In recent years several applications of genetic algorithms have been developed for several protein folding related problems. For example Unger and Moult (1993) have demonstrated the effectiveness of GA in a simple lattice model of protein folding. Recently Pedersen and Moult (1997) used GA to fold small fragments of real proteins. It seems as if structural biology problems in which the overall structure is composed of correctly predicted smaller sub-structures are especially suitable for GA applications.

In order to follow the algorithm developed in this paper, the following basic steps of Genetic Algorithms are sketched in Fig. 2.

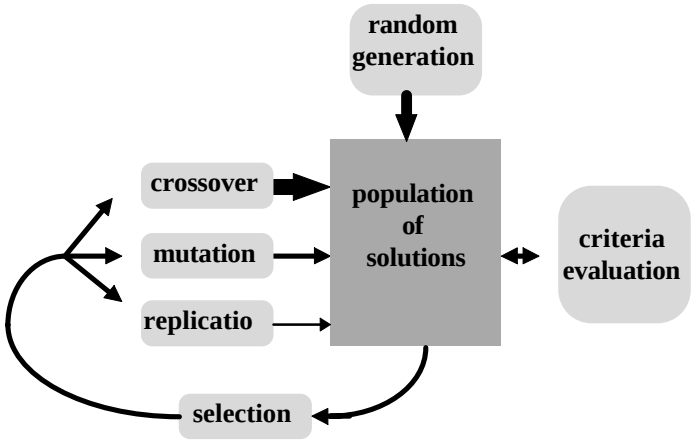


Fig. 2. Basic structure of the genetic algorithm

Representing threading as a Genetic Algorithm string. One of the most important steps in using Genetic Algorithms is the representation of the solutions as strings. This is usually done by binary strings using the alphabet of {0,1}. Since we have chosen to use fixed length strings to represent the individual solutions, we used an alphabet of numbers {0,1,2,3,...,*K*} to represent the solutions (*K* is the sequence length).

Each "1" in the string represents a residue from the sequence aligned consecutively to a position in the structure. Each "0" represents that no residue is aligned to that position in the structure (structure deletion). Number N greater than "1" represents the number of residues that are not aligned to any structure position (N-1 sequence residues are skipped). An example is shown in fig. 3.

In this form of representation it is convenient to perform genetic operations like mutations and cross-over and to validate the generated solutions. The total sum of the numbers that appear in the representation string has to be equal to the threaded sequence length, while the length of the string is equal to the length of the structure. This fact can be used to validate the solution strings.

The fitness function is a normalized energy value, such that solutions with lower energy have higher fitness value and are more likely to be chosen for the genetic operations.

Mutations are performed by increasing randomly the value of a number and offsetting it by decreasing the same amount in other positions. Cross-overs are performed by choosing randomly a position and building two new offsprings by concatenation of the prefix of one, up to the chosen position, to the suffix of the other one. Each offspring is validated and if its sum does not match the sequence length anymore, then a random position is chosen near the crossover point and its value is changed accordingly.

Preventing early convergence. One of the common problems in using Genetic Algorithms is the early convergence of the population to one high fitness individual, which makes it meaningless to continue the genetic process. Since most of the time the population converges to a local minima, it is desirable to maintain high diversity in the population. One possible way of doing it is by creating new solutions with high mutation rates. Another possible solution is, not to allow creation of a solution if it already appears too many times in the population.

Our proposal is to use tree techniques to avoid early convergence of population of Genetic Algorithms. By using the trie data structure to store the solution strings it is possible to make efficient string comparisons and thus to quickly eliminate redundant solutions. In addition, the information stored in the trie can be used later as a convergence indicator of the population.

The Trie Data Structure. A trie is a set of strings to be stored and updated, and allows membership queries [Sedgewick, 1988]. Naively, comparison to detected redundant solutions might need N*N string comparisons of length M, where M is the string length (total N^2M operations). But if we store the strings in a trie data structure there is only a need for N*M string comparisons for fixed size alphabet and O(N(M+K)) for our case where the size of the alphabet can reach the length K of the sequence.

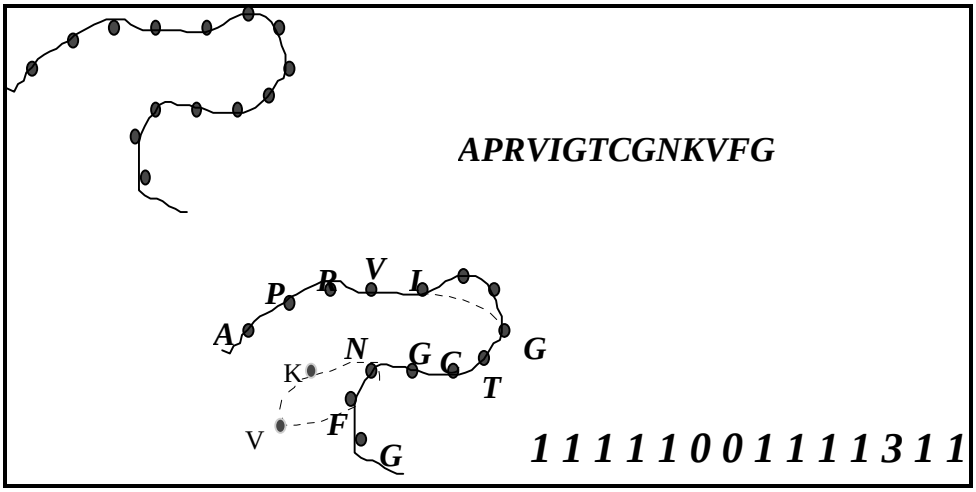


Fig 3. Representing threading as a GA string. Fitting a sequence on a structure to form the threading in the bottom. This particular fit is based on deleting two structure positions and "looping out" two sequence residues. This fit is represented by the string in the bottom, where 0's represent structure deletions, and numbers greater than 1 represent sequence deletions.

Results

We first present data from running the algorithm with the standard core elements definitions, and than present data for full, constraint-free, alignments. The energy potential of Bryant and Lawerence was used throughout this study [Bryant and Lawerence, 1993].

GA compared with enumeration

It is possible to calculate the optimal solution for small search spaces. Here we compare the performance of Genetic Algorithm with full enumeration of all possible threads.

We show here (Table 1) the alignment of the sequence 2mhr (of length 118) and the sequence of 2hmg (of length 175) with the structure of 2hmq (of length 113).

In both cases the same solution was found by the GA and the full enumeration. In the case of 2mhr, full enumeration required 169,911 evaluations of energy function to find the global minimum energy, where the GA required only 20,000 evaluations to find the same result.

For all small examples in which we could run full enumeration, the GA came up with the same optimal solution. We also ran one larger example, 2hmg sequence on 2hmq structure (table 1), in which full enumeration required about 40,000,000 evaluations compared with the same solution found in less than 1% of the time by the GA.

GA results for self threading

We used self threading as another preleminary test of the method. We tested the energy function [Bryant and Lawerence, 1993] to see whether it can detect the native fold when a sequence is threaded onto its own structure. In each starting solution the loop lengths were chosen randomly and the question was, is it possible to get the native loop lengths.

The results for self threading of 1mcp and 2hmq are shown in table 2.a, 2.b. There was no pre-defined limit for loop lengths.

Enumeration	**2mhr**	**2hmg**
structure length (n)	113	113
sequence length (k)	118	175
total core length (c)	92	92
total loop length (k-c)	118 - 92 =26	175 - 92 = 83
number of loops (m)	6	6
search space = $\frac{(m-1+(k-c))!}{(m-1)!(k-c)!}$ *(eq.2.)*	***169,911***	***39,175,752***
Genetic Algorithms	**2mhr**	**2hmg**
number of generations (g)	200	2000
population size (p)	100	100
string length = structure length	113	113
number of evaluations for GA = p * g	***100 * 200 = 20,000***	***100 * 2000 = 200,000***
Enumeration / GA	***~ 10***	***~ 100***

Table 1. GA performance compared with enumeration

1mcp 220										
core definition	3-6	10-13	18-25	38-44	49-55	58-60	68-73	76-81	89-96	102-112
self threading	1	10	18	37	49	57	68	76	88	98
error	-2	0	0	-1	0	-1	0	0	-1	-4

Table 2.a

2hmq 113					
core definition	2-14	18-37	41-61	69-86	90-109
self threading	2	18	41	69	90
error	0	0	0	0	0

Table 2.b

These two tables show self threading of 1mcp (2.a) and 2hmq (2.b). The first line indicates the native position of the core elements. The second line indicates where the first residue of each core element was positioned by the GA. The "error" line shows the difference between the native and computed positions. For 2hmq we found the native structure by finding the optimal threading. For 1mcp the native structure was not found, but actually the energy of the optimal threading found by GA was lower than the native one.

mutation %	population size	number of generations	# evaluations performed	success %
25	50	240	240*50=12000	80
25	100	120	120*100=12000	100
25	200	60	60*200=12000	80
25	300	40	40*300=12000	90
25	400	30	30*400=12000	70

Table 3. Population size vs. number of generations for a fixed mutation rate. Results are averaged over 10 runs.

These two cases were examined by Bryant and Lawerence (1993). For 2hmq they found the same result. For 1mcp they got different threading result, since they had to fix some of the loop lengths because of computational limitations. The current GA threading has lower energy than the previously reported one. Note the (well known) conclusion that the current status of score functions is not good enough since threading experiments find easily alignments that yield "better" threading than the native.

Choosing the best parameters for GA

Usually Genetic Algorithms are sensitive to parameters like mutation rate, population size, number of generations, etc. Here these parameters were examined using the example of threading the sequence of 2hmr on the structure of 2hmq.

The trade-off between population size and number of generation was tested and the results are shown in Table 3. The advantage of using large populations is clear but this trend has a limit. Increasing the size of the population further is counter productive.

mutation %	population size	number of generations	# evaluations performed	success %	# evaluations needed to find the optimum
05	300	40	12000	20	38*300 = 11400
10	300	40	12000	60	31*300 = 9300
15	300	40	12000	80	31*300 = 9300
25	300	40	12000	90	22*300 = 6600
30	300	40	12000	100	20*300 = 6000
35	300	40	12000	80	27*300 = 8100

Table 4. Changing mutation rates, mutation rate refers to the percent of solutions that have been changed. The number of bits changed in each mutation is small. The results were averaged over 10 runs.

mutation %	without trie			using trie		
	# optimal solution found	# second best solution found	# other solutions found	# optimal solution found	# second best solution found	# other solutions found
25	2	16	6	7	13	4
15	2	12	10	2	10	12
3	0	6	18	2	6	16

Table 5. Results of 24 threading of 2hmg sequence on 2hmq structure with different mutation rates with and without trie. The number of times (out of 24) the best , second best and other solutions were found is recorded. Other parameters: population size = 200, number of generations = 600.

Next, the optimal mutation rate was tested. The results (Table 4) show that it is better to use higher rate of mutations to achieve good results, but not to use too high mutation rates. (The peak is around 30%.) This is reasonable since a low mutation rate "freezes" the system while a high rate might not provide enough stability in the population to promote good solutions.

Using trie to speed up the GA (getting results in fewer generations)

As described before early convergence is a real problem in GA. In our application we allowed dynamic control of mutation rates: When the solution strings became too similar we increased the mutation rate temporarily. After few generations the mutation rate was changed back to its original value. The second method for preventing early convergence is a reproduction control method. To implement reproduction control we used the trie data-structure which enables comparing strings with minimal number of operations. Thus we can check efficiently that a new solution does not appear more than a limited number of times in the new generation.

Table 5 shows the results of threading the sequence of 2hmg (of length 175) on the structure of 2hmq (of length 113).

This table shows that using a trie to prevent early convergence of the population is useful. Here we limited the size of each group in the population to be 10% of the population size. A solution that appears already more than 10% in the new generation is rejected and a new solution is selected instead. A limit of 150 trials to find an acceptable solution was set. It was found that the procedure was always able to find an acceptable solution within this limit.

Threading without using core elements

When core segments constraints are not used in the threading process, insertions and deletions are allowed in the whole protein structure. Thus the search spaces becomes much bigger than before and enumeration becomes unfeasible. This creates a significant problem in evaluating the performance of the algorithm, since the optimal solution can not be known. Remember that due to limitations of score functions, the native structure can not be considered as the optimal result.

Table 6 shows that without core constraints the benefit of using GA is dramatic. There is a need to show that indeed optimal or near optimal results can be found in such a huge search space. As discussed above, no proof can be given here, but we present the following indications: Table 7 shows the best final strings of different runs for 1crn self threading with their energy value. Even in this huge search space, some of the best results have been found more than once. These solutions are very similar to each other, although each run started with very different initial populations, which indicates that they are in the neighborhood of the optimal minimum.

Table 8 shows the final strings of the four best runs for 2mhr-2hmq threading with their energy value. Here each one of the top solutions appears only once, but again the fact that these solutions are so close to each other although each run started with totally random population suggest that they are around the optimal minimum.

Enumeration	**1crn**	**2mhr**
structure length (n)	46 (1crn itself)	113 (2hmq)
sequence length (k)	46	118
search space = $\frac{(m-1+(k-c))!}{(m-1)!(k-c)!}$ *(eq.2)*	***2.056 * 10^26***	***~ 2 * 10^69***
Genetic Algorithms	**1crn**	**2mhr**
number of generations (g)	3000	3000
population size (p)	1000	1000
string length = structure length	46	113
number of evaluations for GA = p * g	***1000 * 3000 = 3.0 * 10^6***	***1000 * 3000 = 3.0 * 10^6***
Enumeration / GA	***~ 10^20***	***~ 10^63***

Table 6. GA performance without core constraints.

string	energy	# occurrences
11131113101111011011111001123111110011111111120	-117.14	2
11131113101111011001110111123111110011111111120	-117.22	1
11131111211111010000112111123111100111111111120	-117.44	5
21311112101111010000112111123111111001111111102	-122.18	1
21311112101111021111101001113111311000011111102	-122.24	1

Table 7. The five best solutions, represented as strings, for 1crn self threading without core constraints. Parameters: mutation rate = 25%, population size = 1000, generations = 3000, gap penalty = 1, 20 runs were performed.

string	energy
10022111111111111111112011112112101121101111110101121231111211011021111211111112101112111131011011112101211101	-181.08
11102111111111111111112012012112101111111110111101121231111210111021111211111101121111211131011011112101101211	-181.31
10022111111111111111112012012112101011011031111110112123111111111102111121111110112111121111111111111210110121	-182.10
10022111111111111111112012012112101111111011111101121114111210111021111211111311210111101111111111111210110121	-183.18

Table 8. The four best solution represented as strings for 2mhr-2hmq threading without core constraints. The same parameter set as in table 7 was used.

Discussion

As far as we know this is the first reported attempt to use Genetic Algorithms for the challenge of threading. The search for the optimal thread is a difficult computational problem, which is one of the bottlenecks in using threading as a structure prediction method. Because of computational limitations most of the current threading algorithms have to compromise and solve only a limited version of the actual problem. Pre-defining core elements and allowing insertions and deletions only in loop regions outside these core elements is the most common such compromise. Failures in finding good alignments are often related to such rigid limitations.

In contrast to other methods, our GA threader does not have to use core element definitions. The representation presented here was designed to enable full freedom in choosing positions for insertion and deletions.

It is difficult to prove the usefulness of a threading alignment procedure without building a full threading package that includes threading potential, library of folds, normalization procedure etc. Thus, we consider the work presented here as a first indication that threading by Genetic Algorithm is possible, the next step would be indeed building a full package around this alignment "engine".

Since the energy function we used was defined for residues of core elements only, it may not be suitable for threading without core limitations. This may be the reason why self threading without core segments give results with lower energy than native. Threading needs "gap penalty" definition for deletions and insertions, which is one of the less studied subjects in threading.

The algorithm we used here has a problem of local convergence. It can find the area of the optimal solution (as indicated by the similarity of solutions in tables 6 and 7) but an identical solution is not found in each run. For example in Table 7, one can notice that the third best solution is found more frequently than the best solution. A local optimization to find the minimum energy could be suggested as an additional final step of the process.

The preleminary results presenrted here are encouraging, more experiments are underway to prove the ability of Genetic Algorithms to improve the performance of threading procedures.

References

Bernstein, F. C.; Koetzle, T. F.; Williams, G. J. B.; Meyer, E. F.; Brice, M. D.; Rodgers, J. R.; Kennard, O.; Shimanouchi, T. and Tasumi, M. 1977. The protein data bank, a computer-based archival file for macromolecular structures. *J Mol Biol* 112:535-542.

Bryant, S. H. and Lawrence, C. E. 1993. An empirical Energy function for threading protein sequence through folding motif. *Proteins* 16:92-112.

Bryant, S. H.; Madej, T. and Gilbert J. 1995. Threading a database of protein cores. *Proteins* 23:356-369.

Bryant, S. H. and Altschul S. F. 1995. Statistics of sequence-structure threading. *Curr opin struct biol* 5:236-244.

Bryant, S. H. 1996. Evaluation of threading specificity and Accuracy. *Proteins* 26:172-185.

Chothia, C. 1992. One thousand families for the molecular biologist. *Nature* 357:543-544.

Goldberg, D. E. 1989. *Genetic Algorithms in search, optimization and machine learning*. Reading, Mass: Addison-Wesley.

Holland, J. 1975. *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Jones, D. T.; Taylor, W. R. and Thornton, J. M. 1992. A new approach to protein fold recognition. *Nature* 358:86-89.

Lathrop, R. H. 1994. The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Eng* 7:1059-1068.

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

Lemer, M. R.; Rooman, M. J.; Wodak, S. J. 1995. Protein structure prediction by threading methods: evaluation of current techniques. *Proteins* 23:337-355.

Pedersen, T. and Moult, J. 1997. Protein folding simulations with genetic algorithms and a detailed molecular description. *J Mol Biol* 269:240-259.

Sedgewick, R. 1988. Algorithms. Reading, Mass: Addison-Wesley.

Sippl, M. J. 1995. Knowledge-based potentials for proteins. *Curr Opin Struct Biol* 5:229-235.

Sippl, M. J. 1993. Boltzmann's principle, knowledge based mean fields and protein folding. An approach to the computational determination of protein structures. *J Comput Aided Mol Des* 7:473-501.

Sippl, M. J. 1990. Calculation of conformational ensembles from potentials of mean force. an approach to the knowledge-based prediction of local structures in globular proteins. *J Mol Biol* 213:859-883.

Taylor, W. R. and Orengo, C. A. 1989. Protein structure alignment. *J Mol Biol* 208:1-22.

Unger, R. and Moult, J. 1993. Genetic algorithms for protein folding simulations. *J Mol Biol* 231:75-81.

Unger, R. and Moult, J. 1993. Finding the lowest free energy conformation of a protein is an NP-hard problem: Proof and implications. *Bulletin of Mathematical Biol* 55:1183-1198.