Protein Structure Prediction by Applying an Evolutionary Algorithm

Richard O. Day, Gary B. Lamont
Dept of Electrical Engineering
Graduate School of Engineering & Management
Air Force Institute of Technology
WPAFB (Dayton) OH, 45433, USA
(Richard.Day,Gary.Lamont)@afit.edu

Ruth Pachter
Air Force Research Laboratory
Materials & Manufacturing Directorate
WPAFB (Dayton) OH, 45433-7702, USA
Ruth.Pachter@wpafb.af.mil

Abstract

Interest in protein structure prediction is wide-spread, and has been previously addressed using evolutionary algorithms, such as the Simple genetic algorithm (GA), messy GA (mga), fast messy GA (fmGA), and Linkage Learning GA (LLGA). However, past research used off the shelf software such as GENOCOP, GENESIS, and mGA. In this study we report results of a modified fmGA, which is found to be "good" at finding semi-optimal solutions in a reasonable time. Our study focuses on tuning this fmGA in an attempt to improve the effectiveness and efficiency of the algorithm in solving a protein structure and in finding better ways to identify secondary structures. Problem definition, protein model representation, mapping to algorithm domain, tool selection modifications and conducted experiments are discussed[1].

1. Introduction

Protein structure prediction is a Grand Challenge problem [4, 16]. Solving this problem involves finding a methodology that can consistently and correctly determine the configuration of a folded protein *without regard to the folding process*. The problem is simply stated; however, solving is intractable [15]. Thus, a variety of algorithmic approaches have been proposed [18][20], ranging from GAs, SA, to hybrids between deterministic and stochastic methodologies using nonlinear optimization techniques and maximum likelihood approaches [10], recently reviewed [3]. In this paper we focus on modifications to the fmGA, such as multiobjective implementation of the fmGA (MOfmGA), integrated per residue Ramachandran plots, and a farming model for the parallel fmGA (pfmGA) to improve on previous GA applications for protein structure determination.

All GAs discussed in this investigation utilize the same CHARMm (version C22) energy model as a fitness function[17]. The protein structure is determined by minimizing the energy fitness function. A choice between real and binary values is required. In the past both of these encodings yielded similar results. Thus, a binary encoding was chosen, and the angles discretetized into 1024 (1 MB or 2^{10}) sections for every 360°.

2 Genetic Algorithms

The first step in adapting a model to be solved using a GA (complexity estimates are give in Table 1) is to transform the problem domain solution variable structure into a fixed length binary string – a so called, chromosome. Individual elements of a chromosome are features that correspond to the genes of a chromosome. Feature values are the values that one feature may take on - these represent alleles of a gene. The set of every allele is the genetic alphabet [12]. After a discretized encoding scheme is applied to the problem, the fitness function is evaluated in order to give an indicator if one chromosome is better than another.

2.1 fast messy GA

Following our previous sGA and mGA work, the fmGA (Figure 1) was to be named our GA of choice ([8], [7], [6] and [9]), having lower complexity.

The mGA's advantage over the sGA is in its ability to explicitly create tightly linked building blocks for defeating deception by insuring that there is a good solution in the population of building blocks created in the initialization phase. However, it is extremely expensive to build every combination of a particular building block size to put

[1]The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

Table 1. Complexity Estimates for serial GAs

Phase	*sGA*[a]	*ssGA*[b]	*mGA*	*fmGA*
Initialization	$O(l^n)$	$O(l^n)$	$O(l^k)$	$O(l)$
Recombination	$O(g * n * q)$	$O(g)$		
Primordial	$O(0)$		$O(0)$	$O(l^2)$[c]
Juxtapositional			$O(l \log l)$	$O(l \log l)$
Overall mGA	$O(l^n)$	$O(l^n)$	$O(l^k)$	$O(l^2)$

[a] l is the length of chromosome, n is the size of population, q is group size for tournament selection, g is the number of generations.
[b] l is the length of chromosome, n is the size of population , g is the number of generations of reproduction.
[c] Building Block Filtering

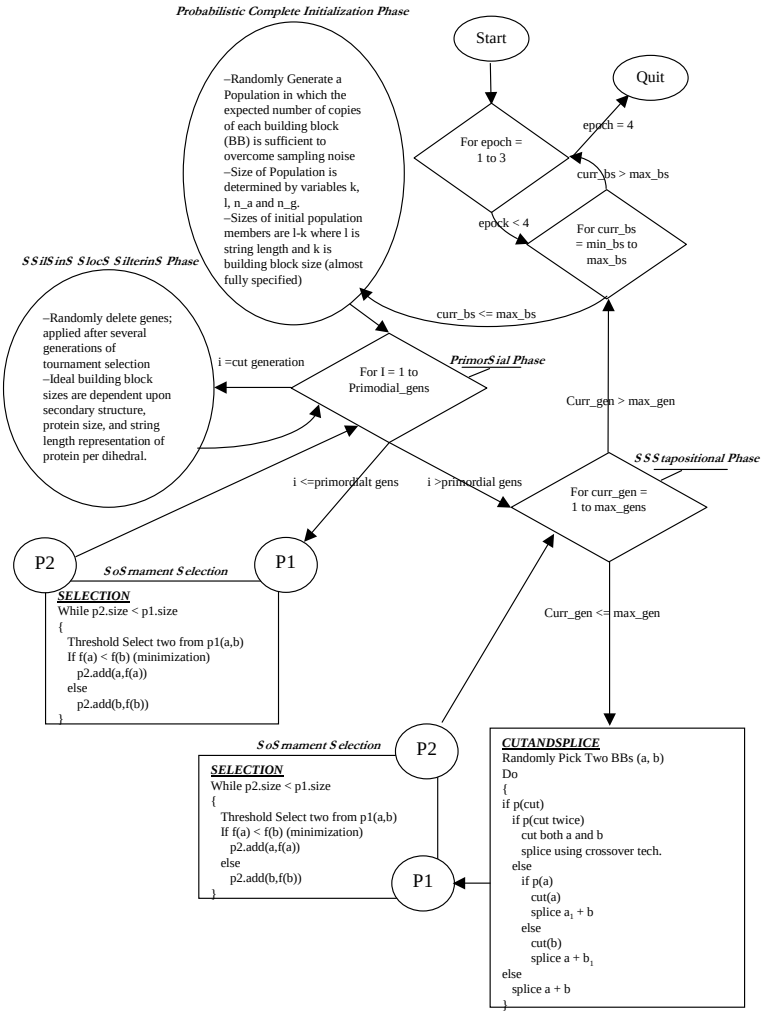

Figure 1. Flow of fmGA.

into a population. The fmGA is designed to reduce this complexity by replacing the initialization phase and primordial phase with a probabilistic complete initialization (PCI) and primordial phase, consisting of selection and building block filtering (BBF). PCI and BBF are an alternate means to providing the juxtaposition phase with highly fit building blocks [13].

The PCI phase creates an initial pop-pool size of n as described by Equation 1, which is probabilistically equivalent to the pop-pool size at the end of the primordial phase of

mGAs.

The population size is the multiplication of three terms from the equations of: the gene-wise probability, the allele-wise combinatoric, and the building block evaluation noise equation [13]. Furthermore, it can be shown that the probability gene-wise equation is the probability of selecting a gene combination of size k in a string of length l' having the total number of genes, l, as given as Equation 2. If, m_g, is assigned to the *inverse* of Equation 2, it is suggested that each subpopulation of size n_g have one needed string, on average, gene combination of size k. Equation 3 defines n_g. If we expect to have one of our needed gene combinations for one particular building block of size k, then we can further claim that we require the needed gene combination for each and every possible combination of k building block size, which makes for 2^k allelic combinations or allele-wise combinatoric population size multiplier. A second multiplier is then defined in Equation 4 (the building block evaluation noise equation). This equation makes for a population size calculation where the selection error between two competing building blocks is no more than α different. Finally, we have a simple, more manageable, population sizing calculation in Equation 5. [13]

$$2^k \frac{\binom{l}{l'}}{\binom{l-k}{l'-k}} 2c(\alpha)\beta^2(m-1) \tag{1}$$

$$p(l', k, l) = \frac{\binom{l-k}{l'-k}}{\binom{l}{l'}} \tag{2}$$

$$n_g = \frac{1}{\frac{\binom{l-k}{l'-k}}{\binom{l}{l'}}} \tag{3}$$

$$n_a = 2c(\alpha)\beta^2(m-1) \tag{4}$$

$$n = n_a n_g \tag{5}$$

Once the population size is determined, the initial population is created and the algorithm begins. The length of the strings, l', is set to $l-k$. The primordial phase performs several tournament selection generations to build up copies of highly fit strings followed by BBF to reduce the string length toward the building block size k. An example of the population sizing calculation can be found in [5]. To conclude, instead of having a huge initialization cost as we do with the mGA, the fmGA has allowed a more optimal initial population mechanism that is statistically equivalent to that of the mGA.

2.2 Parallel fast messy GA

The pfmGA is an extension of the fmGA [13] and is a binary, population based, stochastic approach that exploits Building Blocks (BB)s within the population to find solutions to optimization problems. Our pfmGA may be executed in a single program single data (SPSD) or a single program multiple data (SPMD) mode. The parallelization of this algorithm is based on the Message Passing Interface (MPI) constructs. The pfmGA consists of three phases of operation: the Initialization, Building Block Filtering, and Juxtapositional Phases, all using synchronous MPI based communications. The pfmGA operates independently on each of the processors with communications occurring during the Initialization and Juxtapositional phases (Independent mode).

In the Initialization phase, a population of individuals is randomly generated on each processor. Subsequently, the population members are evaluated. A CT is also generated on each processor *a priori*. The CT is a locally optimized and used for calculating the fitness value of partial strings in the later phases of the algorithm.

The BBF Phase follows and extracts BBs from the population for the generation of solutions. This process occurs through a random deletion of bits from each of the population members alternated with tournament selection. A BBF schedule is provided *a priori* to specify the generations for the deletion to occur, the number of bits to be deleted from each population member and the generations to complete tournament selection. This phase completes once the length of the population members' chromosomes have been reduced to a predetermined BB size. In order to evaluate these BBs ("under-specified" strings), throughout the phase a competitive template is utilized to fill in the missing allele values. These population members are referred to as "under-specified" since each locus position does not have an associated allele value. The BBF process is alternated with tournament selection to keep only the strings with the best building blocks found, or those with the best fitness value around for later processing.

The Juxtapositional phase follows and uses the building blocks found through the BBF phase and recombination operators to create population members that become fully specified (all loci values have corresponding allele values) by the end of the phase. Again the competitive template is used anytime a population member is missing a locus and in the case of "over-specification", where a specific locus is assigned an allelic value multiple times, the first value encountered is the one recorded. At the end of the Juxtapositional phase, the best population member found across all of the processors becomes the new competitive template on each processor. At this point the BB size is incremented and each of the three phases are executed again. After all of the specified BB sizes are executed, the best solution found is recorded and presented to the user.

2.3 Multiobjective fmGA (MOfmGA)

A modified Multiobjective fmGA (MOfmGA) executes using the same basic algorithm structure as the fmGA. The differences include the use of a multiple competitive template design where each objective function is *assigned* a competitive template. This competitive template evolves to "optimize" that particular objective function. Each population member is overlayed onto this competitive template before evaluation of the objective function. As the Juxtapositional Phase completes, population members (after overlaying onto a competitive template if necessary) are written to a file for processing and extraction of pareto front points. Finally, after storing the overall best chromosome into the next competitive template, a PDB file is generated.

3 Design of Computational Experiments

The fmGA is programmed to run in serial and parallel on the following computer systems: Pile of PCs (PPCs), Cluster of Workstations (COWs) and Networks of Workstations (NOWs). The clusters of computers used in this investigation are defined in [5]. The algorithm's generic performance metric is two fold - *goodness* of the structure (effectiveness) and the time to converge (efficiency).

Next we discuss the study of both effectiveness and efficiency for the fmGA when used to determine a protein structure. Specific studies include competitive template generation, a building block size experiment, a Ramachandran constraint experiment and a multiobjective experiment. Efficiency is tested using a Farming Model. Finally, the "goodness" of solutions are evaluated using RMS differences. In this preliminary study we chose $polyalanine_{14}$ (POLY) as a test peptide.

3.1 Competitive Template Generation

The fmGA explicitly manipulates BBs in search of the global optimum and uses the idea of speciation through successive phases of the algorithm. The fmGA uses a competitive template, which is a fully specified population member, to evaluate these partially defined strings or building blocks. By focusing on modifying the process that the fmGA uses to create and update the competitive template during the execution of the algorithm the algorithm's effectiveness is increased.

3.2 Building Block Experiment

The BB analysis is performed in an attempt to identify the building block sizes that result in finding better solutions. A BB is a partial string representing bits from one, some, or all of the dihedral angles that each chromosome represents. The BBs are not restricted to be contiguous bits from the chromosomes but instead can be non-contiguous bits from the chromosome.

The BB analysis conducted covers a variety of BB sizes and compares the results to determine which size produces the best statistical results. One expects a BB size of 35 bits to yield the best due to the alpha helical [3] structure of POLY, known to have 3.5 residues per turn [2].

3.3 Constraints Based on Ramachandran Maps

Search algorithms having constrains on search space by a feasibility function statistically, overtime, must find better solutions. This premise also applies to this experiment, by constraining the search space to have only feasible solutions it is expected that better solutions are found.

3.4 Multobjective Experiment

In the single objective implementation of the fmGA, the CHARMm energy function was utilized and consists of a summation of several terms. In the multiobjective approach, the objectives are drawn from each of the terms within the CHARMm energy function, defined in terms of bonded and non-bonded interactions.

3.5 Farming Model Experiment

Alternate efficiency models, such as the island model, have been previously applied in parallelizing GAs. Due to our energy fitness function calculation, the addition of a farming model is proposed as discussed.

The Component Under Test for efficiency is the fitness function calculation. The farming out of a computationally expensive fitness evaluation should realize speed up in efficiency without affecting the effectiveness. Wall clock time is measured by system clock time to complete (seconds).

4 Results and Discussion

4.1 Multiple Competitive Templates

The multiple CT experiment is our first design modification to the fmGA. This modification requires the fmGA to have the ability to compute a panmetic competitive template[4] in addition to having multiple competitive templates present during computational search. Statistical techniques used are the Kruskal-Wallis (KW) [21] and t-test for paired and unpaired observations (PO) [14].

[4] A panmetic competitive template is derived from the existing multiple competitive templates.

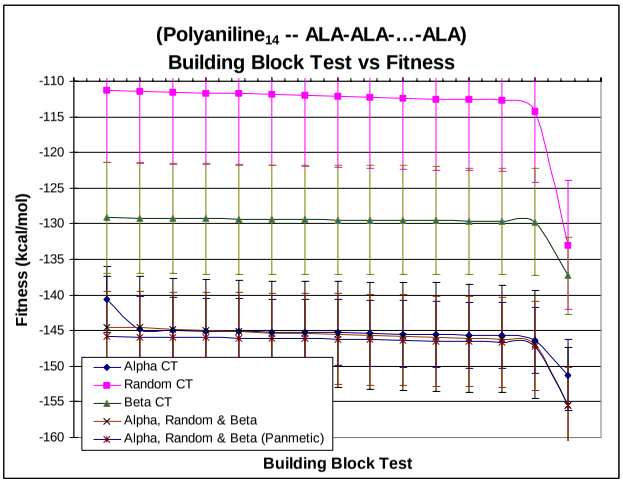

Figure 2. Building Block Test vs. Fitness plot of results for an experiment using multiple methods of competitive template generation on the protein POLY.

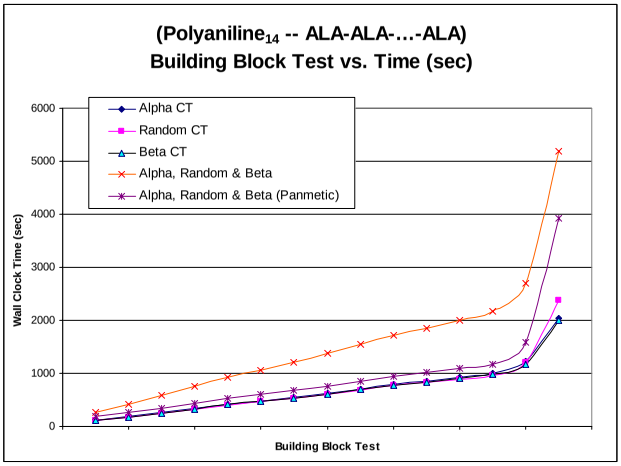

Figure 3. Building Block Test vs. Time to run experiment using multiple methods of competitive template generation on the protein POLY.

Generation of the alpha-helix produces good results; however, the multiple and panmetic competitive template methods also performed well. Both the beta-sheet and randomly generated competitive template generation approaches performed worst. This is illustrated in Figure 2. There is a clear difference between the random, beta-sheet, and all alpha-helix related templates (alpha-helix, multiple, and panmetic competitive template). Similar results are reported with the PO test. Accordingly, the PO test concluded that the order from best to worst is: panmetic, multiple, alpha-helix, beta-sheet, and randomly generated competitive template method. Finally, the KW test also confirmed that the alpha-helix related, beta-sheet, and randomly generated competitive template methods are different. It concluded conceptually that it was 84% confident that these are different (Chi-squared distribution with 2 degrees of freedom and 3.65 quantile) and computationally it is 100% confident that these are different (Chi-squared distribution with 2 degrees of freedom and 924 quantile). A further KW test was conducted on the three alpha-helix related competitive template methods. Conceptually, we conclude that there was no difference between the three (94% confident using df=2 and 0.12 quantile of the Chi-squared distribution). Moreover, computationally the KW test shows with 45% confidence that these are the same. The KW is more strict regarding differences; therefore, we conclude the three competitive template methods are the same statistically.

4.2 Farming Model Experiment

The pfmGA utilizes an island model paradigm to conduct parallel communications between processors. At each stage of the communications, all of the processors communicate their best found population member to processor 0. Processor 0 then determines which is the "best" and communicates that population member back to all of the processors who then update their CT. After the update, all of the processors continue to execute the algorithm independently with independent population members until the next update communication is necessary.

Due to the complexities associated with the energy fitness function calculation, the addition of a farming model *in combination with the island model* is proposed. Farming out the fitness calculations to another set of slave processors allows for a decrease in the overall processing time as long as the computation time is greater than the communications time required. As the slave processors calculate fitness values the masters can do the same or conduct other computations. In addition to speedup gained for the peptide selected in this investigation, the addition of these slave processors allows for the MOfmGA to handle larger proteins.

With the addition of farms, the program becomes multiple program multiple data (MPMD). In the single data model, the GAs execute in parallel and generate populations separately from one another, with interactions only when a migration of a good population members occurs with some probability. While in shared memory machine, a Global population model may be more appropriate since communication cost may not be as much of a concern. In such

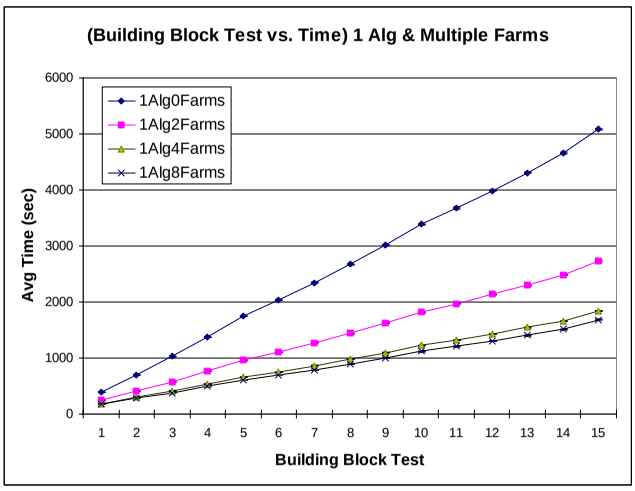

Figure 4. One Algorithm node experiment.

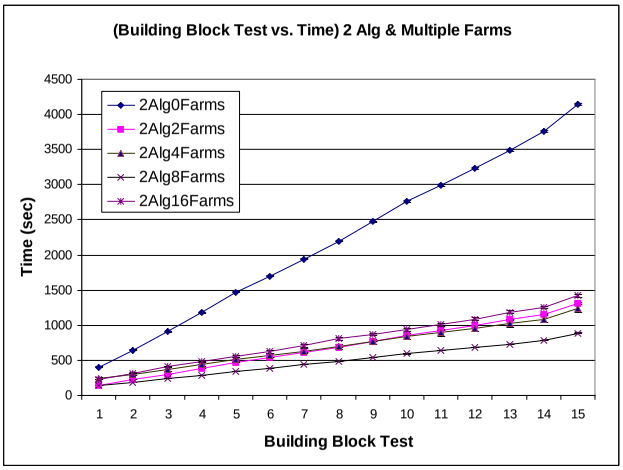

Figure 5. Two Algorithm node experiment.

a shared memory MPMD setup, data does not need to be transferred among processors and data pipelining can be utilized more readily. The systems used in these experiments are not currently shared memory systems but the communication cost is later shown to be insignificant compared to the cost of the energy fitness function evaluations.

The following pfmGA parameters are kept constant (set at standard values) throughout all of the testing: string length = 560 bits, cut probability = 0.02, splice probability = 1.00, primordial generations = 200, juxtapositional generations = 100, total generations = 300. An input schedule is also used to specify during which generations BBF occurs. Computer systems used in this experiment are listed in [5].

A goal of this testing is to determine the speedup associated with increasing the number of farming processors per Algorithm node in the pfmGA. Figure 4 illustrates a plot of one Algorithm node with a number of different farming nodes. Each BB test point represents the average value for a specific BB size (in increasing order) executed by the pfmGA. As the BB size increases, the average execution time also increases. Additionally, Figure 4 and Figure 5 show that as the number of farming processors increases, the average execution time decreases for any given BB test. In this test there exists a significant improvement in modifying the number of farming nodes from 0 to 2 and from 2 to 4. An increase in the farming nodes from 4 to 8 provides a small improvement. The best speedup obtained was with 8 farming nodes where the serial time was 5080 seconds while the parallel time was 1684 seconds yielding a speedup of 3 times. This validates our model and we can draw a conclusion that this model increases the efficiency of the fmGA.

Figure 5 presents the results for two Algorithm nodes tested with a number of different farming nodes. In this testing, the overall population size across all of the algorithm nodes is equivalent to the population size used in the test of a single algorithm node. The serial time was 4140 seconds and the parallel time with 4 compute nodes per farm took 887 seconds yielding a speedup of 4.7.

4.3 Building Block Size Analysis

The BB analysis is performed in an attempt to identify the building block sizes that result in finding better solutions for POLY. A BB is a partial string representing bits from one, some, or all of the dihedral angles that each chromosome represents [19]. The BBs are not restricted to be contiguous bits from the chromosomes but instead can be non-contiguous bits from the chromosome. This analysis covers a variety of BB sizes and compares the results to determine which size produces the best statistical results. The BB ranges chosen for testing included: 16-18, 18-20, 20-22, . . ., and 38-40.

The results of the BB size experiment show that BB sizes of 30-32 yielded the best results for POLY. Although, this BB size is specific for POLY, it should apply to other proteins having an alpha helical structure. Additionally, BB size 30-32 yielded the best overall fitness value found during all of the BB testing of -140 kcal, which is in the neighborhood of the accepted CHARMm fitness for this protein.

4.4 Protein 3D File Generation

A good conformation for POLY was found during the MOfmGA experiment (-170 kcal/mol), which is in good agreement with an alpha-helix (Figure 7).

Figure 6. Found POLY structure with -169 kcal/mol energy value.

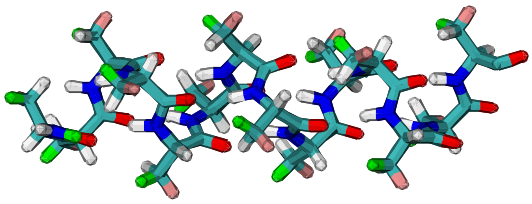

Figure 7. Conformation from a PDB file representing the accepted conformation for Polyalanine$_{16}$

4.5 Ramachandran Experiment

The Ramachandran experiment is conducted to take advantage of problem domain information in restricting the search space (not size) for the algorithm. In the preliminary results the MOfmGA was executed three times for each of the methods to provide statistical results. All results presented here are averaged over three runs. The following MOfmGA parameters are kept constant; cut probability = 0.02, splice probability = 1.00, primordial generations = 200, juxtapositional generations = 200, total generations = 400. An input schedule was used to specify sizes of the building blocks the algorithm uses and during which generations BBF occurs. Tests were conducted using only POLY, with 560 bit length strings and BB sizes 20-24. Furthermore, the n_a variable is set at 100 and, for efficiency of getting results, only a single objective and a single randomly generated competitive template is employed. Figure 8 illustrates the results of the Ramachandran experiment. Clearly, we note that both Ramachandran constraints achieve better results. The mapping cost of using Ramachandran plots is three times that of a non Ramachandran implementation. This is presented in Figure 9. Notice that the mapping for both the Optimistic and Pessimistic implementation takes exactly the same cost in time; therefore, in order to use these constraints, the Pessimistic values are to be utilized because it is statistically more effective and costs the same in time.

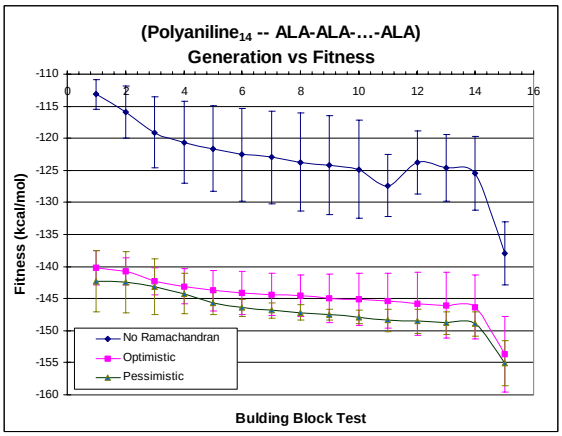

Figure 8. Building Block Test vs. Fitness/Time plots of results for an experiment using no, pesimistic and optimistic Ramachandran plots on the protein POLY. See [5] for the restrictions applied to the landscape for each different method.

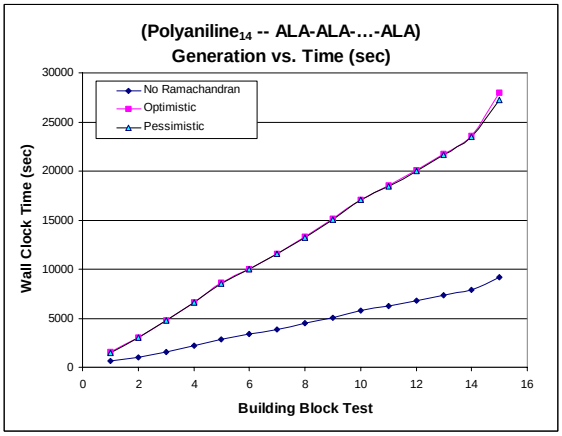

Figure 9. Efficiency results of the Ramachandran experiment.

5 Conclusions

This investigation summarizes our progress of using *MOfmGA*, modified to scale its efficiency to 4.7 times a serial run time. Algorithm development required a major rewrite to prepare for the implementation of the multiobjective approach. The new algorithm has the capabilities to be single and multiple objective and run with single and multiple competitive templates all configurable. The algorithm now provides for optimistic and pessimistic Ramachandran (per residue) constraints and calculation of RMS dihedral and Cartesian coordinate differences from accepted structures. Computational results support our hypothesis that the MO version provides more acceptable results. Overall preliminary results for a poly(alanine) model are encouraging. Future studies will involve beta structures and the villin headpiece [22, 11] as well as participation in CASP [1].

References

[1] www.predictioncenter.unl.gov, 1998-2003.

[2] C. Branden and J. Tooze. Introduction to protein structure. 1991.

[3] J. D. Bryngelson, E. M. Billings, O. G. Mouritsen, J. Hertz, M. H. Jensen, K. Sneppen, and H. Flyvbjerg. *From Interatomic Interactions to Protein Structure*, volume 480, chapter Physics of Biological Systems : From Molecules to Species, pages 80–116. Springer-Verlag New York, 1997.

[4] M. Committee on Physical and E. Sciences. *Grand Challenges 1993: High Performance Computing and Communications*. Office of Science and Technology Policy, 1982.

[5] R. O. Day. A multiobjective approach applied to the protein structure prediction problem. Ms thesis, Air Force Institute of Technology, March 2002. Sponsor: AFRL/Material Directorate.

[6] R. O. Day, J. B. Zydallis, and G. B. Lamont. Competitive template analysis of the fast messy genetic algorithm when applied to the protein structure prediction problem. *ICCN*, page 4, December 22 2001.

[7] R. O. Day, J. B. Zydallis, and G. B. Lamont. Solving the protein structure prediction problem through a multiobjective genetic algorithm. *ICCN*, page 4, December 22 2001.

[8] R. O. Day, J. B. Zydallis, G. B. Lamont, and R. Pachter. Genetic algorithm approach to protein structure prediction with secondary structures. *EUROGEN*, page 6, September 2000.

[9] R. O. Day, J. B. Zydallis, G. B. Lamont, and R. Pachter. Analysis of fine granularity in parallelization and building block sizes of the parallel fast messy ga used on the protein structure prediction problem. *World Congress on Computational Intelligence*, page 6, December 2001. Special Biological area.

[10] J. Ecker, M. Kupferschmid, C. Lawrence, A. Reilly, and A. Scott. An application of nonlinear optimization in molecular biology. *European Journal Of Operational Research*, 138(2):452–458, April 2002. Department of Mathematical Sciences, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180-3590, USA.

[11] A. Fernandez, M. yi Shen, A. Colubri, T. R. Sosnick, S. R. Berry, and K. F. Freed. Large-scale context in protein folding:villin headpiece. *Biochemistry*, 42(3):664–671, 2003. CODEN: BICHAW ISSN: 0006-2960.

[12] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading MA, 1989.

[13] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. pages 56–64, July 1993.

[14] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[15] M. M. L. Khimasia. Np complete problems. http://www.tcm.phy.cam.ac.uk/∼mmlk2/report13/node31.html, 1996.

[16] N. Krasnogor, D. Pelta, P. Mocciola, P. E. M. Lopex, and E. de la Canal. Enhanced evolutionary search of folding using parsed proteins. *Operational Research Symposium*, 1997. Bs. As. Argentina.

[17] G. B. Lamont and L. D. Merkle. Introduction to bioinformatics for computer scientists. Chapter in W. Corne's book, August 2002.

[18] K. Lipkowitz and D. Boyd. *Reviews in Computational Chemistry*, volume 10. VCH Publishers, Inc, 333 7th Avenue, New York, New York, 1997.

[19] S. R. Michaud, J. B. Zydallis, G. Lamont, and R. Pachter. Scaling a genetic algorithm to medium-sized peptides by detecting secondary structures with an analysis of building blocks. In M. Laudon and B. Romanowicz, editors, *Proceedings of the First International Conference on Computational Nanoscience*, pages 29–32, Hilton Head, SC, March 2001.

[20] S. Schulze-Kremer. Genetic algorithms and protein folding. *Methods in Molecular Bilogy*, 143:175–222, 2000. Protein Structure Prediction: Methods and Protocols.

[21] M. R. Spiegel and L. J. Stephens. *Theory and Problems of Statistics*, volume 1 of *3rd*. McGraw-Hill, 1999.

[22] B. Zagrovic, C. D. Snow, M. R. Shirts, and V. S. Pande. Simulation of folding of a small alpha-helical protein in atomistic detail using worldwide-distributed computing. *Journal of Molecular Biology*, 323(5):927–937, 2002. CODEN: JMOBAK ISSN: 0022-2836.