Investigations into a Genetic Algorithm for Protein Sequences

Selwyn-Lloyd McPherson
Biochemistry 218, Winter 2007 – Professor Doug Brutlag

Introduction

In recent years, researchers from many scientific fields have been looking to a range of biological processes to inform new techniques and algorithms for tackling challenging problems in engineering and mathematics. As our understanding of the details of biological phenomena deepens, we are presented with an increasing number of naturally occurring, simple, yet powerful computational methods that exist in nature to perform some useful function. Some of these methods have adapted to serve purposes outside of their natural habitat. Biologically inspired algorithms are beginning to take hold in many areas of study but none has benefited as significantly as the field of optimization – the search across an often complex space for an optimal solution, usually a global maximum or minimum. In cases where brute force or even more conventional optimization techniques fail, these new algorithms offer the promise of finding true global extrema.

Particle Swarm Optimization

One interesting algorithm is Particle Swarm Optimization (PSO) which attempts to model the group movement of insect swarms in order to efficiently traverse a solution space (Kennedy and Eberhart, 1995),. In biology, if one member of a swarm notices a beneficial place or entity (such as protection or food), that member will move towards it. Consequently, this information will be transferred to the rest of the group – the other group members will "notice" and the mean direction of the swarm will be directed towards that item. Eventually, the swarm will arrive at the place of interest. Similarly, we can imagine such a swarm traveling through a virtual solution space in which individuals are attracted to extrema defined by some objective function that describes the space. If virtual particles are given properties similar to their natural counterparts, they, too, will ideally converge at the optimal solution.

PSO is a great example of a successful transition from the world of biology to the world of computation. The algorithm itself is simple; each particle is defined by only two properties – the current position and current velocity. Population-level variables number four to five at most. The method is easily reusable for a variety of different problems, as the particles may exist in a multidimensional space. In addition, because of the semistochastic property of swarms, they are able to explore a space better than a completely deterministic system. At the same time that the group is being pulled towards a particular minimum or maximum, a random component of the particles' velocities (their search trajectories) allows members to explore nearby spaces, decreasing the probability of getting stuck in a local extrema.

Increasing Interest and Success

Of course, PSO is not the only successful biologically inspired algorithm. Ant Colony Optimization, which utilizes the theory of communication networks in ant populations, is being used to solve complex graph theory problems (Dorigo and Gambardella, 1997). Neural networks, which are loosely modeled after cortical structures in the brain, are also proving to be an important addition to many engineering fields including regression analysis, classification and data processing. Neural networks also enjoy wide use in the bioinformatics community where they are used for sequence analysis, classification and alignment (Blekas, et al., 2005; Rost, 1996). In an attempt to take neural networks further and to better simulate their natural counterparts, several projects around the country, including one here at Stanford in the Boahen lab, are aiming to create large silicon brains from anywhere between several thousand neurons to 10^{11} neurons, the approximate number in a human brain (Boahen and Zaghloul, 2006; Izhikevich, 2006). These systems, apart from deepening our understanding of the brain, can actually be used for computation.

One of the most successful biologically inspired algorithms, however, is the Genetic Algorithm (Goldberg, 1989), which has been used with great success to solve large-scale optimization problems where other algorithms often fail.

Genetic Algorithms

A Genetic Algorithm (GA) is a global search heuristic that uses the basics of natural selection at a genetic level to evolve a population of initially poor solutions towards a problem-specific optimal solution. The fundamentals of the algorithm are familiar to those with even a modest biological background – they include *reproduction* (with genetic recombination, i.e. *crossover*), *mutation* and *selection*. While many non-biological fields have borrowed this powerful form of optimization, it has rarely (if ever) been brought back to its native environment, i.e. that of actual genetic information. The goal of the experiments detailed below was to determine if and how GAs could successfully used to operate on protein sequences. Furthermore, because genetic algorithms are commonly plagued by slow and often suboptimal convergence, these experiments are also an investigation into the alleviation of these common problems in the space of genetic sequences.

The Algorithm

Conceptually, the algorithm takes an initial, usually randomized population of poor solutions to an optimization problem and attempts to determine a best solution by utilizing the concept of natural selection to evolve the population. With each new iteration (or generation) of the algorithm, the fitness of the solutions (the individuals) in the population is calculated. Individuals with higher finesses are more likely to "reproduce," spawning similar solutions in the next generation. Like offspring in nature, however, these children are not identical to their parents. Due both to the recombination of the parental genetic material and to random mutation, children are potentially more fit than their parents. Upon the creation of each new generation, the finesses are once again evaluated and the process repeats. In this way, the population theoretically converges on an optimal solution or a close approximation thereof.

The Objective Function and Goal

GAs must have some objective function that not only defines the solution space of the problem but also provides a method for evaluating the fitness of each individual in each generation. In this case, fitness is defined by a protein alignment score as executed by JAligner (Moustafa, 2007), an open-source, Java-based implementation of the Smith-Waterman sequence alignment algorithm (Smith and Waterman, 1981). The Smith-Waterman algorithm is best suited for the purposes of a GA because it performs local alignment, giving a more dynamic evolution than a global alignment algorithm such as the Needleman-Wunsch. The protein selected is of no particular consequence; in this study, a 132 amino acid long retinol-binding protein from the *Bos taurus* was used (NCBI Accession #AB28336). *Fitness* was defined simply as:

$$F(X) = \frac{\text{alignment score}}{\text{perfect score}}$$

where a perfect score was determined by performing a self-alignment. This yields values from 0 to 1.

Ideally, we would like to begin with a population of random individuals and arrive the goal protein. When we consider that this is an *N*-

dimensional problem, where N is the number of amino acids in the protein, the task seems impossible. Luckily, we are aided by the fact that the solutions themselves are not continuous. Of course, this leads to the conclusion that we could simply enumerate every possible sequence of length N and do a multiple alignment to determine which proteins are the closest to the goal. In fact, we already know the optimal solution – it is simply the goal protein! Rather than focus on finding the solution, however, this study investigates *if* and *how* that solution is reached.

Encoding individuals

In the traditional implementation of the algorithm, individuals in the population are represented as one-dimensional "chromosomes" that encode the solution in some problem-specific way. This simplistic formulation is beneficial as it allows for the easy operation of mutation and recombination that can simulate the natural processes off which they are based. Binary bits are often used, but studies using real-valued encodings have shown improvements in speed and accuracy (Davis, 1991; Janikow and Michalewicz, 1991). For many implementations, it can be somewhat of a hurdle to encode a complex solution into a one-dimensional chromosome. As an example, we can suggest a toy problem in which we attempt to minimize the quartic function:

$$f(x,y) = \frac{1}{4}x^4 - \sqrt{7}x^3 + \frac{1}{13}x + 4y^2$$
$$\text{for } -50 \le x \le 50 \quad \text{and} \quad -50 \le y \le 50$$

A chromosome-like encoding the solutions for x and y could be created, using binary digits, such as:

$$\underbrace{\mathbf{10100}}_{x=20}\underbrace{\mathbf{101010}}_{y=42}$$

Unfortunately, using binary digits comes with the complication of variable-length chromosomes, which may necessitate potentially cumbersome bookkeeping measures. Using a real number chromosome encoding of "**2042**" is potentially more helpful, but with so little information to mutate, the solution space becomes too coarse and thus more random, hindering the smooth traversal of the solution space.

Fortunately, by bringing the fundamental concepts of the GA back to their natural environment, these problems are alleviated! Chromosomes are encoded as they would be in nature. An RNA sequence, as opposed to a DNA sequence, is used to ease the translation to from nucleotides to amino acids, though the choice is not important in any other respect. The alphabet used is simply the set of RNA nucleotides, "AGCU." Though the Smith-Waterman algorithm can handle both nucleotide sequences as well as amino acid sequences, due to the obvious length of the nucleotide sequence, performance is much faster with amino acids. Conversely, the methods of random mutation and genetic modification are more natural at a nucleotide level. The best option is to perform all genetic operations on the nucleotides and translate to amino acids solely for the purposes of JAligner.

Selection

After the creation of each generation, the finesses of the individuals are ranked. At this point, there must be some method for picking individuals to "mate" – to exchange genetic material to produce new individuals. Ideally, better-fit individuals will have a higher probability of mating. It is important, however, that some less-fit individuals to reproduce as well; these individuals are often the source of (good) genetic variation. This is particularly true in GAs as populations often become saturated with one chromosome, usually the current best fit. In such homogenous populations, the lack of variation causes evolution to slow dramatically. One main research area in GAs is to determine how to determine and adapt variation efficiently to obtain an optimal solution (Andre, et al., 2001). Good

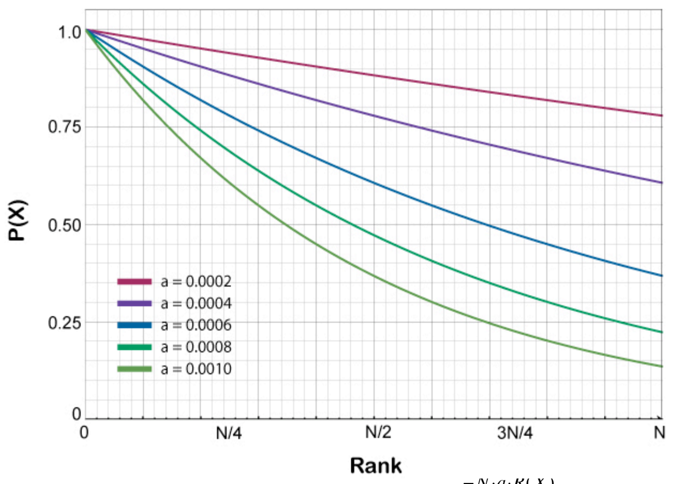

Figure 1. Members of the family $e^{-N \cdot a \cdot R(X)}$ for a = 0.002...0.010. The probability of being picked for mating, P(X), is dependent on the rank of the finesses. Higher values of *a* give provide a higher selection pressure.

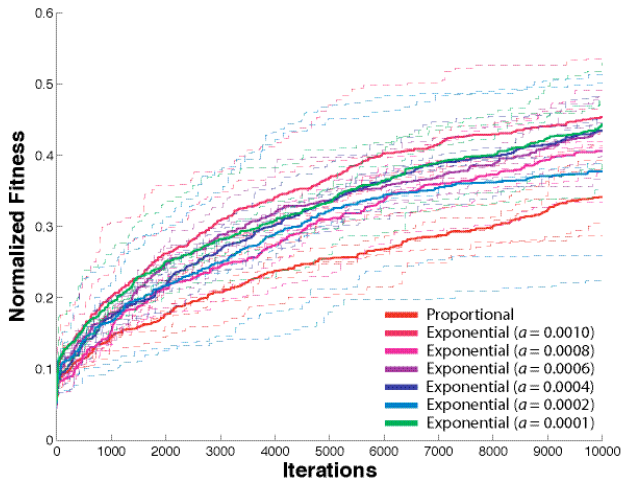

Figure 2. Exponential selection performs better than proportional selection, though the value of *a* appears not to be a factor. This is most likely due to a highly homogenous population, which decreases the impact of ranking and selection. The dotted lines are individual runs and the solid line is the mean of that set.

selection pressure is known to be an important aspect of algorithm performance

Several selection methods have traditionally been used in GAs. Roulette-wheel selection is the most popular. This is essentially a proportional selection where the probability of a chromosome being picked to mate is given by:

$$P(X) = \frac{F(X) + \varepsilon}{\sum_{i \in I} F(X_i)}$$

where *F(X)* is the fitness of chromosome *X*, *I* is the set of all chromosomes in the population, and ε is some small value so as to avoid a near-null probability for lower fit chromosomes. In practice, this can be achieved by creating a vector of size *S* and filling it with instances of the chromosomes such that a chromosome with probability *P*/*S* would be instantiated *P* times. A simple random selection from the vector will provide accurate selection.

Other popular methods include tournament selection (Miller and Goldberg, 1995) and stochastic universal sampling (Baker, 1987), which are essentially variations on proportional selection. More abstract and complicated selection methods have been suggested but here a novel one is presented. An *exponential* selection can be formulated as:

$$P(X) = e^{-N \cdot a \cdot R(X)}$$

where *N* is the number of individuals in the population (here, always 50), *a* is some small value, and *R(X)* is the rank of chromosome *X*, where the chromosome with the best fitness has rank 0, the next with has rank 1, etc. Choosing a value of *a* from 0.001 to 0.0001 gives a good range of selection pressure (Figure 1). This selection method can be implemented easily: simply select some random cutoff from 0 to 1 and then randomly select a chromosome with a probability, *P(X)*, greater than this value. In practice, this method performs better than proportional selection regardless of the value of *a* (Figure 2). Because populations tend to evolve to be rather homogenous, most chromosomes have equal or approximately equal fitness. In the case of proportional selection, this causes the algorithm to degenerate to an essentially random selection choice. Ranking the chromosomes, in most cases, preserves the small differences between chromosomes.

Genetic Operators

In nature, random operation on the genetic code is the source of variation that drives evolution. This includes:

- Point mutation – change in one nucleotide
- Inversions – the change in order of several nucleotides
- Transposition – the introduction of one or several groups of some number of nucleotides
- Deletions – the deletion of several nucleotides

All of the above were incorporated into this implementation of the GA to make it as natural as possible. After two chromosomes have been chosen to mate, each is operated on with a probability of p_{mut} (for each of the above types).

It has been reported that genetic algorithms work most efficiently with mutation rates near $p_{mut} = 0.001$ (0.1%). Of the small values of p_{mut} tested, a mutation probability of 0.0015 (0.15%) was optimal. There was, however, a huge improvement in the success of the algorithm when higher mutation probabilities were tested. A value of 0.1 was optimal for this problem and one run at this mutation probability did converge on the goal protein, proving that genetic algorithms can in fact be successfully applied to the protein sequence problem (Figure 3). There is, however, a limit to p_{mut}; values of 0.3 and 0.6 performed worse than $p_{mut} = 0.1$, though $p_{mut} = 0.3$ did better than smaller values.

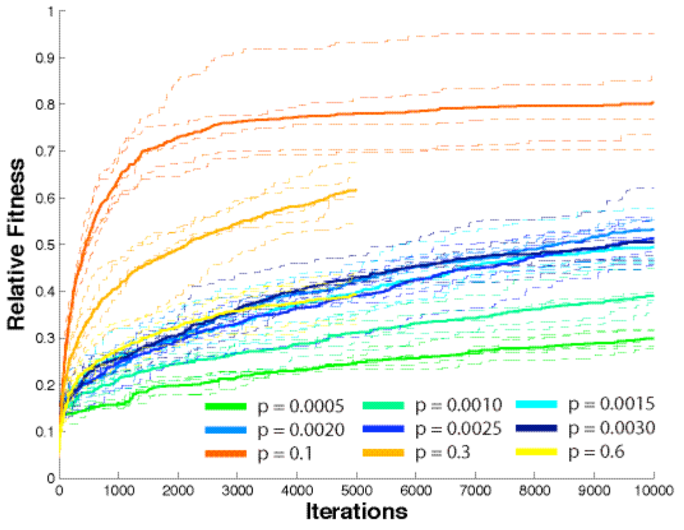

Figure 3. Varying the mutation rate has a significant impact on the success of the algorithm. Here, a run of $p = 0.1$ reached a similarity of 95% and was considered to have converged. Very low mutation rates and very high mutation rates show poor performance. Some evolutions are reported through 5000 iterations due to computational time constraints

These findings are in contrast with the mutation probabilities commonly found in the literature. The argument that a mutation rate of 0.1% mirrors that found in nature is irrelevant as the time course of the GA and of nature are not comparable. It is likely that mutation probabilities are problem-specific and, at least for the case of protein similarity, higher mutation intuitively increases the population diversity and can converge to the optimal solution. Extremely high mutation rates, however, force the algorithm to degenerate into a random search. There is a line between successful and unsuccessful values of p_{mut}, though determining an acceptable range is not easy based on reasoning alone.

Crossover

Crossover is the mechanism by which two new individuals are created from two parental chromosomes. In nature, this action can be simplified to two different types: single and double crossover (Figure 4). Both of these types are simulated in the GA, each with a probability of 0.5. In other implementations, the location of the crossover point can be a concern, as improper crossover may disrupt the encoding of the variables on the chromosome. In these simulations, however, there is no such problem; the chromosome and the information contained therein are not sensitive to the location breaks or crossovers.

Mutation Probability	URPP
0.0005	1.0421
0.0010	1.0564
0.0015	1.0828
0.0020	1.1195
0.0025	1.1234
0.0030	1.1508
0.1	3.6395
0.3	6.7584
0.6	9.1816

Table 1. The average number of unique residues per position (URPP) is a good indication of population homogeneity. Higher mutation rates yield populations that are more diverse

Homogenous Populations

Population homogeneity is a significant problem faced when using genetic algorithms. In the case of protein sequences, homogeneity can be defined as the number of unique amino acids per position across the chromosomes in the population. Initially, the population is random and this value is somewhere near 20. Very often, however, within the first few iterations, the number of unique amino acids decreases dramatically to some relatively stable value for the rest of the simulation. This is generally dependent most importantly on the mutation rate (Table 1).

There is a large focus in the literature on determining effective ways to increase population heterogeneity (Beyer, et al., 2002; Smith, et al., 1993). It is true that homogeneous populations traverse the solution space much more slowly than heterogeneous ones but a diverse population alone will not ensure optimal convergence. Increasing the p_{mut} does increase population diversity but at high rates of mutation, the algorithm converges on a sub-optimal solution.

Simulation Time

GAs are not inherently computationally intensive, but the parameters used and the peculiarities of the problem may cause computation time to be a constraint. In this study, one main limitation was the time required to perform an alignment, which is generally greater than the execution of the rest of the operations in the algorithm. Though one alignment is on the order of fractions of second, an alignment is performed $N \times I$ times per run, where N is the number of individuals in the population and I is the number of iterations in the simulation. The values used here are $N = 50$ and $I = 10000$, equaling 5×10^5 alignments per evolutionary run. Combined with the necessity of performing at least five trials for each experimental condition, time can quickly become a major factor. To exacerbate matters, the Smith-Waterman algorithm is sensitive to the length of the sequence being aligned; the algorithm operates in *O(mn)* time, where *m* and *n* are the length of the sequences being aligned. Though it was possible to perform all of the experiments reported here on a 1.67 GHz Apple PowerBook G4, run times often ran into the tens of hours for a set of experiments.

One obvious solution to the problem of time is more processing power. Code optimization is also an option. There are, however, more "accessible" solutions to improve the speed of the algorithm. To decrease the number of alignments performed, the score of sequences that have been aligned can be saved in a hash table. In relatively homogenous populations (average number of unique amino acids per position in the population $\leq \sim 3$), it is beneficial to avoid repeat alignments. In these cases, searching through a table is faster than doing an alignment and this improvement decreases the running time significantly.

Unfortunately, for populations that are less homogenous (as in the case of a high mutation rate), the hash table becomes filled with so many unique entries that searching through it quickly becomes more intensive than re-aligning a repeat

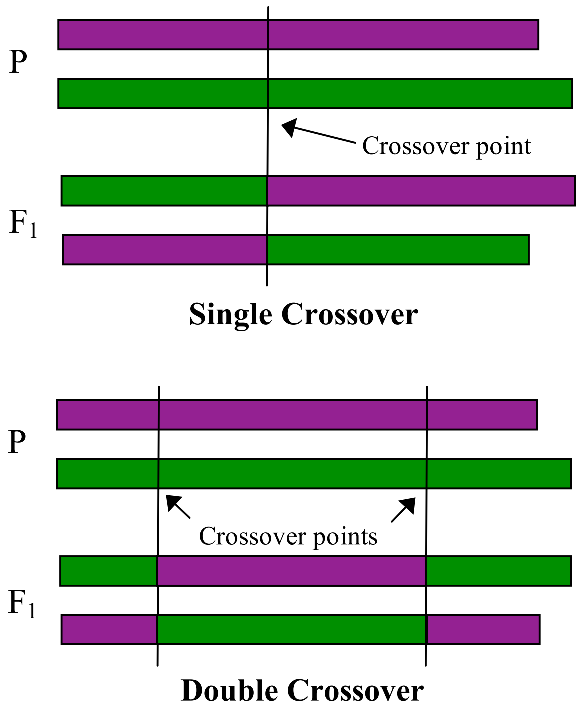

Figure 4. Two types of crossover.

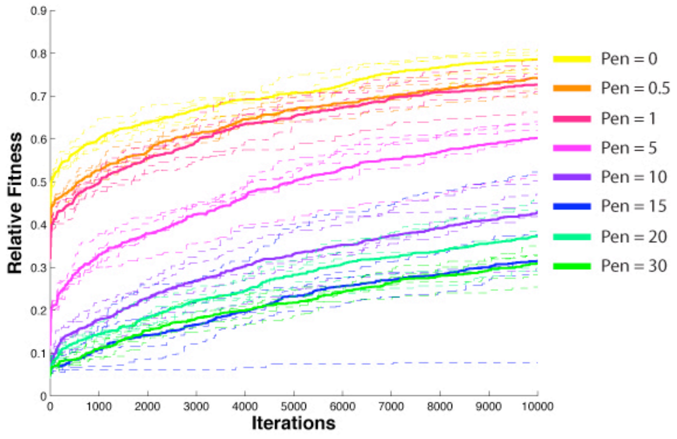

Figure 5. Lower gap penalties yield higher scores. The nature of the alignment, however, differs greatly as the gap size changes. Here, gap creation penalties are reported. Gap extension penalties are 0.1 * Gap Penalty.

sequence. In these cases, the hash table option had to be abandoned. Ideally, these two options would be leveraged dynamically throughout the simulation such that, if the hash table becomes too full, the algorithm resorts back to performing alignments.

Stop Codons

One interesting peculiarity that often arises in a wide search of the protein space is the appearance of stop codons. In order to be true to the nature of translation, stop codons should in fact terminate the protein. This can hinder evolution, especially in homogenous populations (Figure 6, Figure 7). In order to avoid "stunted" evolution, stop codons were instead represented by a placeholder amino acid, *X*, which Jaligner considers to be a mismatch in every case.

Gap Penalty

In addition to the parameters of the algorithm itself, the success of this implementation also depends largely on the properties of the alignment algorithm. One major element of the Smith-Waterman algorithm is a penalty for both creating a gap and extending a gap. Unfortunately, there is no intuitive choice of penalties that is robust for all situations.

Generally, lower gap penalties generally lead to higher scores (Figure 5). There are some intricacies related to the gap penalty, however. As might be expected, solutions obtained by alignment with no gap penalty are not entirely cohesive and do have many gaps. The following is an example of one such gapped yet high scoring alignment that was the result of one evolutionary run:

```
Query    MAATVKGRVRLLNNWDIC-DMVSTFTDTERTAGNADPAKFSVK
alb_bos  MSATAKGRVRLLNNWDVCADMVGTFTDTE------DPAKFKMK

Query    YSWSLASFLRKGNASLLRLLVSKRLGLSAGRRPGPTRGC----
alb_bos  Y-WGVASFLQKGN--------------------------RVPQ

Query    ST--QMETFAVRWSAR-LTNVDG---LND--------D-----
alb_bos  DTDY--ETFAVQYSCRLL-NLDGTCA—DSYSFVFARDPSGFS

Query    P-----------E-C----F-------FTCSD--S-K
alb_bos  PEVQKIVRQRQEELCLARQYRLIPHNGY-C-DGKSER

Score    362.0 / 712.0 = 0.508
```

This can be contrasted the resulting protein from a run with a gap penalty of 30.0:

```
Query    HLILKTSFXRVAVQRSCRLRNTSGTLASSYSFVWERDTSMFGP
         |.|:.|.:...|||.||||.|..||.|.|||||:.||.|.|.|
alb_bos  HWIIDTDYETFAVQYSCRLLNLDGTCADSYSFVFARDPSGFSP

Query    RAEKIVRRRQEEYCAARHYRLAPNSGYASGSSQRQIL
         ..:||||:||||.|.||.|||.|::||..|.|:|.||
alb_bos  EVQKIVRQRQEELCLARQYRLIPHNGYCDGKSERNIL

Score    234.0 / 712.0 = 0.329
```

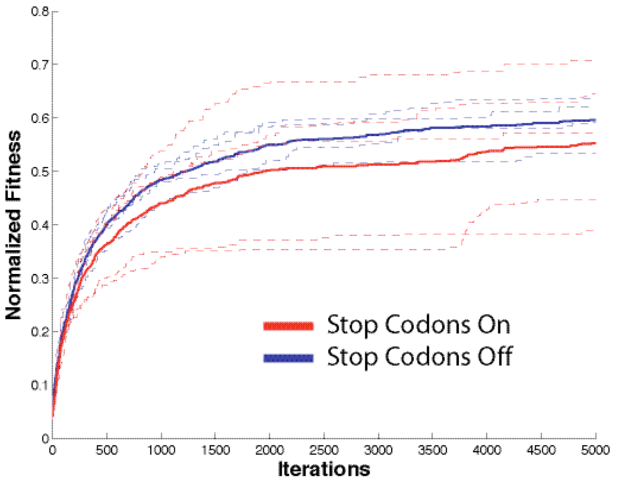

Figure 6. Treating stop codons as real termination signals can stunt evolution. Here, two runs are held back due to a stop codon that has proliferated throughout the population. The dotted lines are individual runs and the solid line is the mean of that set. It is possible, though not certain, that a population may recover from a prolific stop codon (one run here appears to).

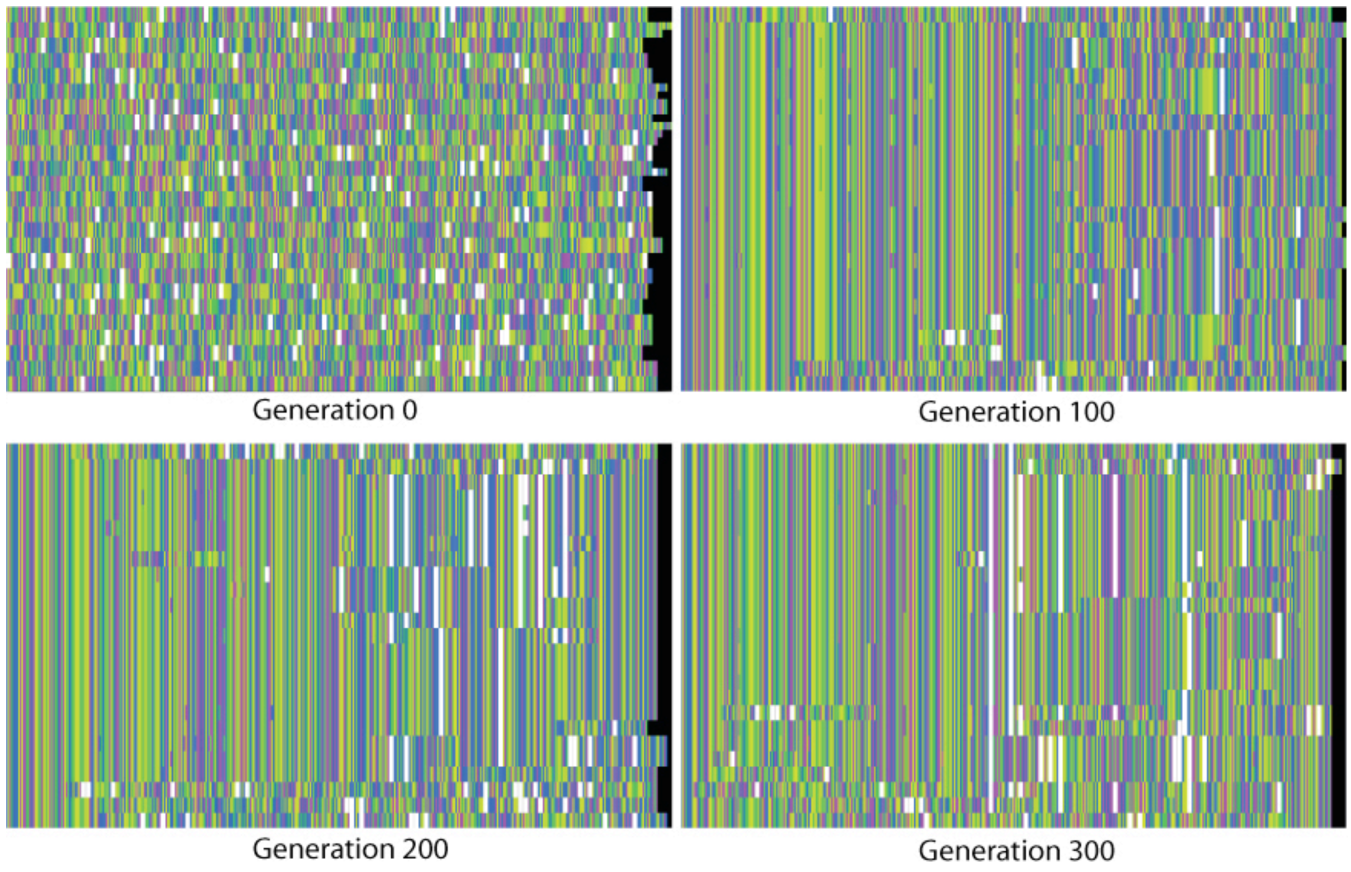

Figure 5. A visualization of a population through early evolution. Nucleotides are each colored with a different color, with the exception of sequences “UAG”, “UGA”, and “UAA” (the stop codons), which are colored in white. Though some variation exists, homogenization is apparent. It this run, a stop codon proliferated across the population. If these codons are allowed to terminate the protein, the evolution is often stunted.

High gap penalties intuitively select against alignments with gaps. Interestingly, though, even the sequences obtained with low gap penalties score higher when high penalties are applied to them than the sequences obtained operating with high penalties. Gap-tolerant alignments are more lenient and thus the algorithm is able to take more of the total sequence into consideration. This allows for the simultaneous optimization of spatially disparate regions of the chromosome.

Given the number of gaps in the alignment with no penalty, however, one could argue this solution seems less correct than the alignment with a high penalty. In fact, though zero gap penalty alignments give high scoring solutions, a gap penalty of zero is discouraged, as there is no consequence for extremely long chromosomes. There must exist a balance, then, between the desire to allow gaps in order to improve evolution and the desire to obtain a good, cohesive solution. This is, in part, a subjective question depending on one’s interpretation of a “good” alignment.

A gap penalty of 9.0 yielded runs that converged (defined as at least 95% similarity) on the goal protein. These alignments were solid across the entire length of the protein and contained few (if any) gaps:

```
Query    MSATAKGRVRLLNSWDVAADMVGTFTDTEDPAKFKLK
         |||||||||||||:|||.|||||||||||||||||:|
alb_bos  MSATAKGRVRLLNNWDVCADMVGTFTDTEDPAKFKMK

Query    YWGVASFLQKGNDDHWIIDTDYETFAVQYSARLLNLD
         ||||||||||||||||||||||||||||||.||||||
alb_bos  YWGVASFLQKGNDDHWIIDTDYETFAVQYSCRLLNLD

Query    GTCADSYSFVFARDPSGFSPEVQKIVRQRQEELCLSR
         |||||||||||||||||||||||||||||||||||:|
alb_bos  GTCADSYSFVFARDPSGFSPEVQKIVRQRQEELCLAR
```

```
Query   QYRLIPHNGYCSGKSERNIL
        |||||||||||·||||||||
alb_bos QYRLIPHNGYCDGKSERNIL

Score   677.0 / 712.0 = 0.951
```

Though a gap penalty of 9.0 allowed the algorithm to converge properly, it is not clear whether other gap penalties would yield similar results. More investigations on the effect of the penalty must be performed for a more complete understanding.

The Future

Here, it is proven that genetic algorithms can in fact be successfully applied to the problem of protein sequence. Exploring the implications and details of this application will yield new insights both about the nature of protein evolution and that of GAs. Some considerations for future work were made apparent by this study:

Although the alignment score was a relatively robust way of evaluating the fitness of chromosomes, perhaps a more holistic method could be applied. A solid technique may require combining some properties of a simple Smith-Waterman alignment with others tailored to the specific problem. Leveraging the number of gaps with the desire to explore a wide range of sequences is important for this goal.

In this study, the BLOSUM62 matrix was used for all alignments. Performance is most certainly dependent on the matrix used and using others may prove beneficial. Substitution matrices are essentially a quantification of the rate of mutation between the amino acids, whether from Dayhoff in the case of the PAM matrices or from Henikoff and Henikoff in the case of the BLOSUM matrices (Dayhoff, et al., 1978; Henikoff and Henikoff, 1992). Greater measures could be taken to incorporate these rates into the actual point mutation rate. In this study, point mutations were done at random; a more accurate method would take into account the amino acid being mutated and determine an appropriate replacement based on the scoring matrix used. Tailoring the mutation rate to the matrix or even creating a new matrix based on some alternate mutation scheme would most likely improve results. Along the same lines, rather than initializing the population with random chromosomes, sequences could be picked that stay true to the natural amino acid distribution.

An investigation of the topology of the solution space may improve the understanding of why certain populations do or do not converge on optimal solutions. Traditionally, suboptimal solutions are due either to homogenous populations or to local extrema. Describing the space for a purely mathematical function is as easy as visualizing the function but for protein sequence some other method of description is necessary. The solutions to the protein alignment problem are discretized, which allows for the full comprehension of the space, but exploring and reasoning about its characteristics and implications is a more complicated task. Do local maxima / minima exist? Is the space monotonic? These questions require a more in-depth analysis of the complete set of solutions to a protein alignment.

Although the path of the evolution towards the goal protein in a genetic algorithm does not exactly replicate the evolution of proteins in nature, some similarities must exist. To simulate the full range of a protein's evolution though time, one could envision a *set* of objective proteins corresponding to the temporal evolution of a real protein. Once a random population evolves towards a protein in a lower organism, the objective function could be changed and the population could be directed towards a protein later in evolution. This could be done for a sequence of several proteins to model the progress from some low-level organism protein to the human version.

In order to make the evolution of proteins in genetic algorithms more like the evolution of real proteins, we must ask what the objective function in nature is. Very often, the fitness of a protein in nature is close to 0 or 1; while some proteins are in fact robust, many are not resilient to changes in amino acid sequence. Unfortunately, the fitness of a protein is unknown until it is allowed to function. One method of improving a

GA in this way would be to inform the algorithm of known functional blocks. A protein with one or more of these blocks would have a high fitness. Chromosomes could be separated into separate genes and might be put into virtual environments with other small molecules. Upon running the simulation, chromosomes that encode for proteins that, together, perform some useful function (like the creation of biomass) would be given a higher fitness. This is a large undertaking but would not be out of the question once the parameters of a genetic algorithm in the protein sequence space have been properly worked out.

Finally, though tinkering with the parameters of an algorithm is useful in gaining an understanding of how it works, this method is time consuming and often unintuitive. A machine-learning algorithm (perhaps another GA!) could be used to automate this process and arrive at a problem-specific set of conditions under which the most optimal solutions are reached. An efficient, general learning algorithm for the purposes of tuning a GA would be of great value to those working with these algorithms.

References

Andre, J., Siarry, P. and Dognon, T. (2001) An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization, *Advances in Engineering Software*, 49-60.

Baker, J.E. (1987) Reducing Bias and Inefficiency in the Selection Algorithm. *Proceedings of the Second International Conference on Genetic Algorithms and their Application* (Hillsdale), 14-21.

Beyer, H.-G., Schwefel, H.-P. and Wegener, I. (2002) How to analyse evolutionary algorithms. University of Dortmund, Dortmund, Germany.

Blekas, K., Fotiadis, D. and Likas, A. (2005) Motif-based Protein Sequence Classification Using Neural Networks, *Journal of Computational Biology*, **12**, 64-82.

Boahen, K. and Zaghloul, K.A. (2006) A silicon retina that reproduces signals in the optic nerve, *Journal of Neural Engineering*, **3**, 257-267.

Davis, L. (1991) *Handbook of genetic algorithms.* Van Nostrand Reinhold, New York.

Dayhoff, M.O., Schwartz, R.M. and Orcutt, B.C. (1978) A Model of Evolutionary Change in Proteins, in *Atlas of Protein Sequence and Structure*.

Dorigo, M. and Gambardella, L.M. (1997) Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transactions on Evolutionary Computation*, **1**, 53-66.

Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning.* Kluwer Academic Publishers, Boston, MA.

Henikoff, S. and Henikoff, J. (1992) Amino Acid Substitution Matrices from Protein Blocks, *PNAS*, **89**, 10915-10919.

Izhikevich, E.M. (2006) Polychronization: Computation with Spikes. *Neural Computation,* **18**, 245-282.

Janikow, C. and Michalewicz, Z. (1991) An experimental comparison of binary and floating point representation in genetic algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Kaufman, M. (ed). San Francisco, 31-36.

Kennedy, J. and Eberhart, R. (1995) Particle swarm optimization, *Proc. of the IEEE Int. Conf. on Neural Networks,* **4**, 1942–1948.

Miller, B.L. and Goldberg, D.E. (1995) Genetic Algorithms, Tournament Selection, and the Effects of Noise. IlliGAL Report No 95006. University of Illinois at Urbana-Champaign, Urbana, IL.

Moustafa, A. (2007) JAligner: Open source Java implementation of Smith-Waterman.

Rost, B. (1996) PHD: predicting 1D protein structure byprofile based neural networks, *Meth. Enzymol,* 525-539.

Smith, R.E., Forrest, S. and Perelson, A.S. (1993) Searching for diverse, cooperative populations with genetic algorithms, *Evolutionary Computation*, **1**, 127-149.

Smith, T.F. and Waterman, M.S. (1981) Identification of common molecular subsequence, *Journal of Molecular Biology*, **147**.