Chapter 2

Michael Levin

Cell and Developmental Biology Dept.
Harvard Medical School
Boston, MA

mlevin@husc8.harvard.edu

Locating Putative Protein Signal Sequences

Abstract

This chapter presents an application of genetic algorithms to a problem in molecular biology. Many proteins occurring in cells participate in biochemical events such as degradation, chemical modification, directional transport, etc. It has been shown that in certain cases, a string of amino acids serves as a specific signal; thus proteins which carry this sequence within their primary structures participate in some molecular event, while proteins lacking this sequence do not (the endoplasmic reticulum retention signal "KDEL" is a good example). Finding the sequence of a specific possible signal based only on the primary structures of a group of proteins thought to carry it is a very difficult task. No good algorithm currently exists for locating brand new signals. A genetic algorithm is described here which is able to discover such sequences. This algorithm is able to search the enormous state space of all possible signals in reasonable time, and locate likely signal sequences (which can then be tested empirically). The algorithm can also be used to find signature sequences in related proteins. Because genetic algorithms are domain independent, a parametrization study is also presented, which shows optimal values of certain constants for this specific task.

2.1 Introduction

Many proteins important in cell function participate in various processes (retention in or targeting to specific organelles, chemical modification, degradation, secretion, etc.). In certain cases, the signal which determines exactly which proteins participate in a given process is a short string of amino acids within the primary structure of the proteins. The endoplasmic retention signal, KDEL, is a good example of this (Pelham, 1990).

So many examples of signals have been found (Bairoch, 1991) that when one has several proteins (called the "in" group), all of which undergo some particular event, it becomes tempting to search for a sequence which might serve as the recognition signal. Once a potential sequence has been found (one that occurs only in that group of proteins), the hypothesis can be tested by artificially grafting the signal onto a protein which doesn't normally participate in the event. If the protein is seen to then undergo the event, the hypothesis is confirmed.

One problem with this process is that given the primary structures of several proteins, it is a very difficult task to come up with a potential signal sequence; if the proteins are of significant length, it is very hard to identify a common (but unique to the group) region by eye, especially since certain amino acid homology rules and groupings may apply. If one has a pretty good idea what this signal might be, simple pattern matching, weight matrix analysis, or discriminant analysis can be used. However, there are no good methods for easily finding a completely new signal.

This problem reduces to the task of finding the longest string which matches optimally somewhere within all members of the functionally-defined "in" group, and does not match a random set of proteins not belonging to the group (the "out" group). A closely related problem is to find a signature sequence which can be used to tell certain proteins apart from similar ones (such as the "A-G-L-x-F-P-V" signature for histone H2B, Wells, 1989). This task thus touches on features of machine learning, pattern recognition, classification systems, and feature abstraction.

Considering the fact that the data is noisy (i.e., one or more of the "in" group proteins may not carry the signal, but participate in the event of interest for other reasons), and the fact that the signal sequence may not be 100% conserved among all proteins, the search space of all possible signals of a given length (usually 3-10 amino acids long) is a very difficult one. If a very fast computer is available, and one is willing to restrict the search to signals less than about six amino acids long, an exhaustive search of all possible short strings may be feasible. However, as the length of the proteins involved grows, and one wants to look at signals which may be somewhat longer, this quickly becomes impractical with respect to the time involved to perform the search (the time required is proportional to 30^N where N is the maximum number of characters in the signal).

A set of algorithms which has recently been shown to be able to find solutions in difficult search spaces are known as "genetic algorithms" (Goldberg, 1989, Davis, 1991, Holland, 1992, Koza, 1992). These domain-independent algorithms simulate evolution by retaining the best of a population of potential solutions and mutating these to arrive at the next generation's population. This process is repeated until a solution of sufficient quality is found (or computational resources are exhausted). The algorithms have proven to be robust and effective for a wide variety of problems, such as symbolic regression, process control, generation of emergent behavior, classification, and pattern recognition (see Koza, 1992 and references therein). GAs have also been used in molecular biology (Dandekar, 1992).

This approach can be used to locate likely candidates for functional protein signals (De La Maza and Tidor, 1992, used a very similar problem to study the effects of Boltzmann selective pressure). This is done by performing random mutation and fitness selection over a population of candidate signal sequences. Each individual in this population is a string of amino acids of some length. Its fitness is proportional to how well this sequence matches the members of the "in" group, and inversely proportional to its match with the "out" group. Genetic algorithms are used, rather than the more general technique of genetic programming because in this case the map from discrete character set genome to the possible solution space is a very natural one. This algorithm is shown to effectively and easily locate potential signal sequences with no initial data other than the functional grouping of proteins and their primary structure. Since the genetic algorithm approach is domain-independent, a parameterization study is performed, to determine the optimal parameters for locating such sequences.

2.2 Implementation

The algorithm is fairly simple, and is easily coded in C. Each member of the population is a string (called a "schema") of some length (minimum length is usually set to 3, maximum length to 10) over the alphabet consisting of the single letter codes for amino acids, plus the symbols _* (the wildcard, or "any amino acid" symbol), a ("an acidic amino acid"), b ("a basic amino acid"), n ("a neutral amino acid"), h ("a hydrophobic amino acid"), p ("a polar amino acid"), and c ("a charged amino acid"). This string can be directly compared to any protein's primary structure. Thus for example, the hypothetical string "TY*Sa" would match a protein which contained a threonine, followed by a tyrosine, followed by any amino acid, followed by a serine, followed by any acidic amino acid. Additional symbols can be added (such as the numbers 0 through 9) which can stand for certain other homologies (for example, 0 may stand for "either S or A here"). The string is the member's genetic material — one chromosome.

Several operators are used. First, a fitness function has to be defined, which can evaluate the worth (i.e., success at differentiating the "in" proteins from the "out" proteins) of any individual. This is returned as a scalar floating point number, which allows unambiguous ordered ranking of all individuals. This number is based on parameters designed to take into account several desirable qualities of a candidate solution; higher numbers indicate better schemata. The fitness function used in this implementation, when applied to a schema S, returns a number which is equal to:

$$k_1.\mathit{knowledge}(S) + k_2.\mathit{size}(S) - k_3.\mathit{vagueness}(S) \qquad (2.1)$$

where knowledge(S) is equal to:

$$match\_ins(S) - k_4.match\_outs(S) \qquad (2.2)$$

match_ins(S) determines how well schema S matches the proteins in the "in" group. *match_outs*(S) does the same for the "outs" group. The degree of match is computed as the sum of all matches to proteins in a group, divided by the number of members in the group. The degree of match to a given protein is defined as the number of matching characters (at the best-matching position within the protein) divided by the length of the schema. The match to the "out" group is subtracted to ensure that the best individuals are those which match the "out" group least.

size(S) determines the effects of the schema size. In general, the value of *size*(S) should be proportional to the length of the schema, because it is better to have the complete signal sequence than a part of it. It may, however, include special nonlinear terms to punish schemata that are too long.

vagueness(S) determines how specific the schema is. It is proportional to the number of non-specific symbols occurring in S (such as * etc.). This term is also subtracted in equation 1 because it is best to have as specific a sequence as possible (while still matching optimally).

The constants k_1 through k_4 are parameters that the user can adjust for specific effects. Normally $k_1 >> k_2$, k_3 because the most important thing is for the schema to differentiate between the "in" group and the "out" group. However, the other terms ensure that if two individuals have similar matching ability, the more specific and longer ones will be considered more fit. The constant k_4 can be changed to control how specific the signal is to the "in" group. It is usually less than 1.0 because even a signal that occurs somewhat in non-belonging proteins can be useful if it always occurs in belonging ones.

Once the most fit members of a population are identified, their genotypes are used to construct the next generation. Two possible operators are mutation, and crossover. A single mutation event performed on a schema S (asexual reproduction), as used in this algorithm, consists of choosing at random among: deleting a symbol at a random position within S, adding a random symbol somewhere within S, or changing a random symbol within S to some other random symbol. Appropriate safeguards are used to ensure that schemata don't become too small or too large. Other than that, the mutation is completely random, with no foresight as to the effects on its performance. Crossover consists of picking two individuals, and producing two new ones by swapping random parts of the parents' genome.

Crossover was used in the initial trials of these experiments, but resulted in premature convergence of the population on suboptimal solutions (data not shown). Thus, all results shown in this chapter utilize simple mutation only. These results are consistent with those of Fogel and Atmar, 1990, who conclude that complex genetic operators such as crossover and regional inversion do not compare favorably with simple mutation (unlike Holland, 1975 and Koza, 1992, who claim that crossover produces better results than mutation).

The control flow of the algorithm is shown in Figure 2.1. After the parameters are set, the "in" and "out" groups are read in from disk. The "out" group should ideally consist of proteins which are related to the proteins in the "in" group, but known (from empirical evidence) not to participate in whatever event functionally defines the "in" group. Alternatively, the "out" group can consist of randomly chosen proteins, or even of random sequences of amino acid symbols.

An initial random population is then created. The population size is the parameter P_1 — this is what determines how many solutions the algorithm is working with at any time. The bigger the value of P_1, the longer it takes to evaluate each generation; however, higher values of P_1 make it more likely that a good solution will be found. Typical values of P_1 can be from 300 to 1000. The user can, at this point, seed the initial population with several initial guesses. This can be used to improve a guess obtained by other means, or to help speed up the search when some of the signal is known, but it is not a good practice in general because it can cause the search to prematurely converge on some solution and ignore one which may turn out to be better.

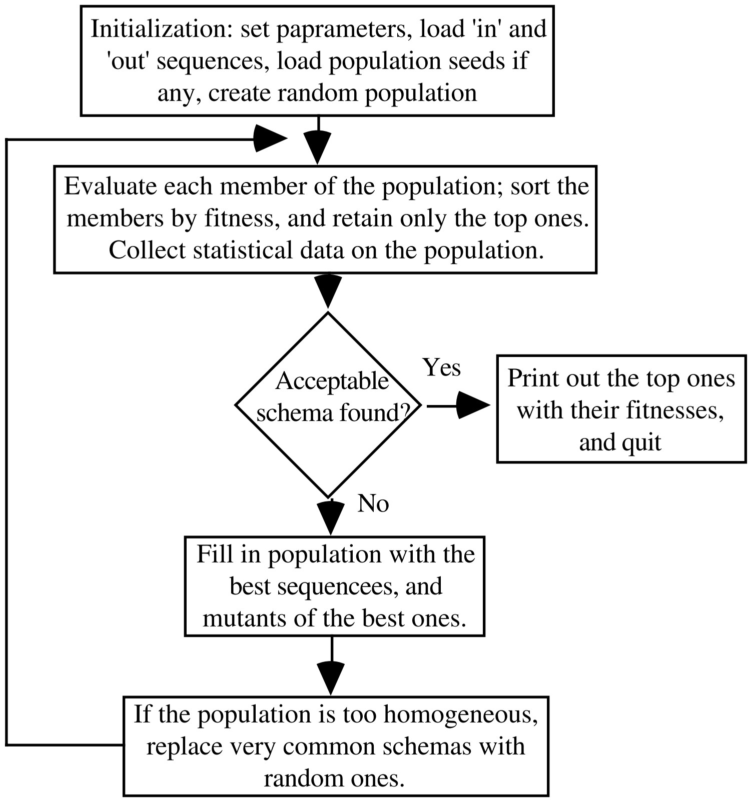
Initialization: set paprameters, load 'in' and 'out' sequences, load population seeds if any, create random population
Evaluate each member of the population; sort the members by fitness, and retain only the top ones. Collect statistical data on the population.
Acceptable schema found?
Yes
Print out the top ones with their fitnesses, and quit
No
Fill in population with the best sequencees, and mutants of the best ones.
If the population is too homogeneous, replace very common schemas with random ones.

Figure 2.1: The genetic algorithm flow of control.

Then, each individual is evaluated according to the fitness function, and the top P_2 schemas are chosen. P_2 is usually between 10% and 70% of P_1. Too high a value of P_2 results in slow convergence, while too low a value may cause premature convergence due to early elimination of potentially good schemata. It is important to note that evaluating the fitness of a given individual is the most computation-intensive step in this algorithm. As the population homogeneity begins to rise, a simple trick can be used to cut down the evaluation time (which can be critical, when the "in" group is large). This method takes advantage of the fact that if more than one schema in the population have identical sequences, only one has to be evaluated, and its fitness can be assigned to all of them. Thus, previous to evaluating fitnesses, the population is sorted by alphabetical order. For each schema S_n (n>l) if it is identical to S_{n-1}, then the fitness assigned to S_n is simply copied from S_{n-1}; otherwise, the fitness of S_n is calculated explicitly.

The population is then rebuilt, to consist of mutated copies of the best individuals, as well as unchanged versions of these individuals. This is "elitist" selection, and ensures that good schemas are never lost from the population. This process continues until either an acceptable solution is found, or the time limit expires. This process can contain several additional features. For example, if the population homogeneity becomes too high, some copies of the most frequent individuals can be replaced with random schemas, or mutated heavily in an attempt to inject variety into the system (the elitist selection ensures that this cannot decrease the maximum fitness found in the population). The whole algorithm is summarized by the following pseudocode:

```
1. Read initial data — in and out groups, parameters N, P, Q, R, S, etc. Place protein sequence in two-dimensional string array

2. Build up a random population of schemas, or read them in from a file. Place sequences into two-dimensional string array. Crossover and mutation are accomplished as string operations (i.e., character and substring substitutions, deletions, inversions, etc.) on the members of these arrays.

3. Until top fitness is acceptable, or allotted time has expired, do:

    A. Computer fitness for each member of the population, by matching its sequence to each member of the in and out groups. Fitness is calculated as in Eq. 1 above, using simple string matching.

    B. Sort the population. Leave the top $n$ members unchanged. Set the next $P$ members to strings which arise from crossovers between randomly chosen members of the $N$ best. The choice is biased to favor crossovers between dissimilar schemata.

    C. Introduce $q$ mutations into the members resulting from crossover, and set the remainder of the population to consist of mutated versions of the top $N$ members.
```

D. Compute and plot the top fitness, average fitness, and homogeneity of population as a function of generation number.

E. Compute total homogeneity of population. If this is higher than an acceptable level *R*, then eliminate all but one copy of each individual, and fill in the rest of the population with crossovers between the remaining individuals and random schemata.

4. Print out the top *S* non-identical schemata, their fitnesses, and their locations within each member of the in group.

In this algorithm, the computational complexity (as measured in the number of string comparisons per generation) as a function of total protein lengths is O(n). That is, it increases only linearly with increases in the number of total amino acids in the in and out groups. However, the total time spent on the search is not necessarily O(n) because different numbers of total generations are required to find adequate solutions for different sets of proteins, and because of the stochastic nature of the algorithm.

2.3 Results of Sample Applications

This algorithm was tested on many different kinds of signals (data not shown). Two examples are illustrated here in detail. Figures 2.2 and 2.3 show the progress of the search over time (in generations on the abscissa). Three quantities (explained below) are monitored; their magnitude is normalized between 0 and 1 (on the ordinate).

The first sample application illustrates how the algorithm finds the KDEL signal (given the sequences of the following proteins found in the GenEMBL database): *H. vulgare* GRP94 homologue, rat immunoglobulin heavy chain binding protein (BiP), rat calreticulin, and rat protein disulfide isomerase (accession numbers X67960, M14050, X53363, X02918, respectively). For this run, the parameters are set as shown in column 1 of Table 2.1. The results of the run are seen in Figure 2.2. The KDEL sequence is found in 64 generations, which represents about 2.5 hours of real time on a lightly loaded (average system load during run = 1.05) DecStation 5000 workstation. Interestingly (perhaps), it initially found other sequences common to these proteins (with 100% fit to each): "EED" and "EEEa". Once a schema has been found, and determined (empirically) not to be of interest, others can be searched for by entering this sequence into the "out" group (to ensure that the search disregards it).

Parameter	KDEL	Histone signture	Default
Population size (P_1)	800	800	800
Survival size (P_2)	300	300	30%
K_1	15	15	15
K_2	2	2	2
K_3	3	3	3
K_4	0.1	0.8	0.1

Table 2.1: Parameter settings for various runs.

The second sample application illustrates how this method can be used to find signature sequences. In this case, the histone H2A signature *A-G-L-x-F-P-V* (Wells, 1989) can be found by running H2A variants in the "in" group and the H2B, H3, and H4 proteins in the "out" group. In this experiment, the "in" group consisted of sea urchin *(P. miliaris)* late histone H2A-2, human histone H2A gene (lambda-HHG55), *P. miliaris* histone H2A-2.1 gene, and the murine H2A gene (accession numbers M11085, K01889, M14140, X16495, respectively). The "out" group consisted of sea urchin *(P. miliaris)* late histone H2B-2, *P. miliaris* histone H2B-2.2 gene, *P. miliaris* gene for histone H3, chicken histone H3 gene, *A. thaliana* histone H3 gene, *X. laevis* histone H4-I gene, and the newt histone H4 gene (accession numbers M11088, M14143, VO1140, J00869, M35387, M23776, M23777, J00954, respectively). For this, the specificity constant needs to be higher than usual. The constants in this experiment are given in column 2 of Table 2.1. Figure 2.3 shows that the H2A signature is found at generation 206. Interestingly, another one is found, which is considered by the algorithm to be even better (because it doesn't contain any non-specific characters): LQFPVGR at generation 84.

Several interesting things can be noted from these sample runs. The solid line shows the fitness of the best individual at each generation. This curve is monotonic, since the elitist selection ensures the best individuals are never lost. In these and some other runs (data not shown) the maximum fitness curve is reminiscent of the punctuated equilibrium hypothesis (Eldredge, 1985) — long stretches of little change interrupted by sharp improvements. This may be due to the fact that the mutation rate used here is too low to cause changes in top fitness over small time periods.

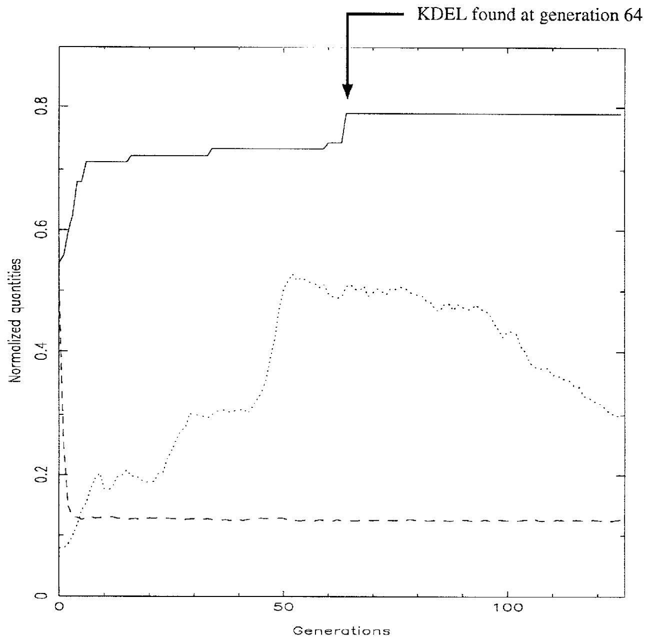

Figure 2.2: Locating the KDEL sequence.

A genetic algorithm search was performed with the parameters given in column 1 of Table 2.1. The solid line represents the fitness of the most fit schema at any generation. The dashed line represents the average length of the schemata at a given generation. The dotted line represents the homogeneity of the population.

The maximum fitness of the second plot starts out much lower than that of plot 2, since the target string of experiment 2 is more complex, and the average fitness of a random individual is likely to be lower. The dashed line representing the average length of all schemata drops quickly to the optimal length. This is somewhat surprising since the length constant in the fitness function is low, and it might be expected that the length not be important (and thus not be selected for) until the fitness becomes quite high and the population converges. The dotted line represents the population homogeneity (as computed by taking the sum of the average similarities of each individual to all others in the population). Interestingly, it is non-monotonic and complex; this is an emergent phenomenon — there is nothing in the fitness function to directly cause such a curve. Note that these curves are very different between Figures 2.2 and 2.3, suggesting that the large-scale population dynamics are different for different instances of this search problem.

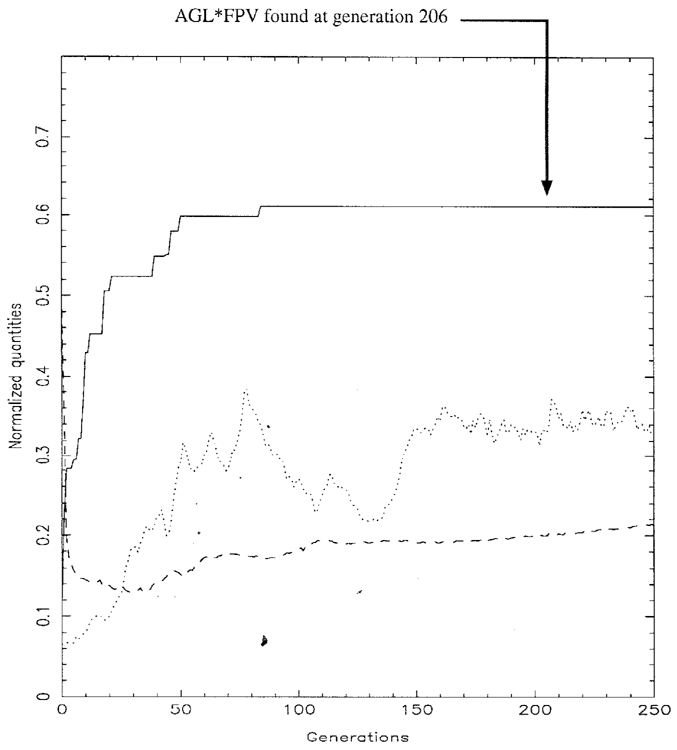

Figure 2.3: Locating the Histone H2A Signature Sequence.

A genetic algorithm search was performed with the parameters given in column 1 of Table 2.1. The solid line represents the fitness of the most fit schema at any generation. The dashed line represents the average length of the schemata at a given generation. The dotted line represents the homogeneity of the population.

2.4 Parametrization study

With so many variables in this domain-independent algorithm, it becomes interesting to: 1) determine what combination of settings are optimal for the protein signal problem, and 2) examine the properties of the algorithm as they vary with the parameters. For these purposes, a parametrization study was performed. In all of these studies, the dependent variable was the generation number in which the desired answer first appeared ("generation of discovery"). The problem set in all cases was to locate the KDEL sequence (using half-lengths of the proteins given above). All parameters except the one being changed are set to the values in column 3 of Table 2.1. A study of variation (since the algorithm is a non-deterministic one) was performed; 20 repetitions of exactly the same problem and parameters showed that differences in generation of discovery were of the range ±13 (data not shown). This is to be considered as the significant difference level for the experiments described below. In all of the figures, the value shown is the average of 10 repeat runs.

The first part of this study examined the dependence of the algorithm's efficiency in finding the KDEL sequence on the size of the population used. A population size of 400 found the solution in 41 generations, while a population size of 1400 found the solution in only 17 generations. Intermediate values of population size produced intermediate values of generation of discovery. Populations of sizes 300 and smaller did not tend to locate the solution at all (within 2000 generations). Figure 2.4 summarizes the dependence of generation of discovery on the size of the population. Clearly it is better to use larger generation sizes. However, since larger generation sizes also take longer to evaluate, it is interesting to examine how the time of discovery relates to the generation size. It is important to note that these times are relative (because they depend on what kind of computer the tests are run on).

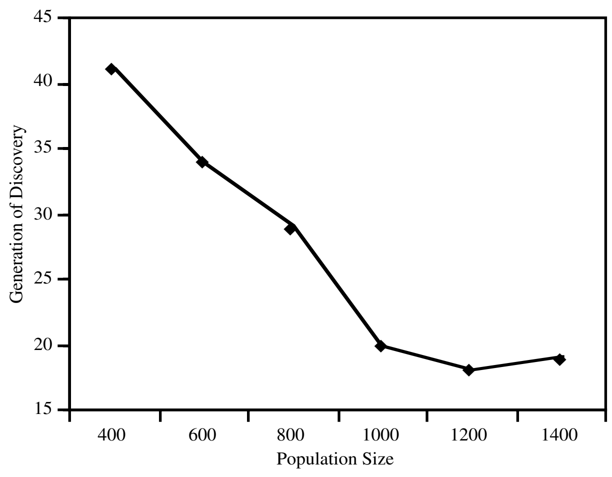

Figure 2.4. Dependence of solution rate on population size.

A series of genetic search algorithms was performed on the KDEL problem, each using a different population size (given on the X axis). The other parameters are set as in the third column of Table 2.1. The generation number at which the KDEL sequence was found is plotted on the Y axis.

A population of size 400 found the solution in 58 minutes, while a population of size 1400 found it in 80 minutes. Intermediate population sizes produced intermediate results. Figure 2.5 summarizes this data, showing a U-shaped relationship. For small generation sizes, it takes longer to find the solution because of the large number of generations necessary. For large population sizes, it also takes longer, because of the computational cost of evaluating large populations. The optimal value seems to be about 800, which allows the solution to be found in just 31 minutes.

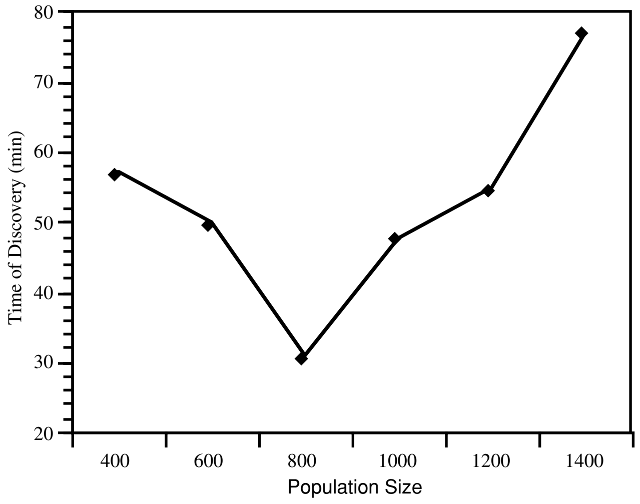

Figure 2.5: Dependence of Time to Discovery on Population Size.
A series of genetic search algorithms was performed on the KDEL problem, each using a different population size (given on the X axis). The other parameters are set as in the third column of Table 2.1. The time (in minutes) at which the KDEL sequence was found (relative to start time) is plotted on the Y axis.

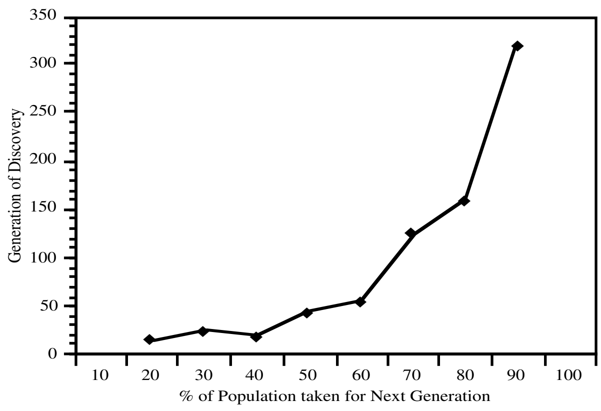

Figure 2.6: Dependence of Generation of Discovery on Survival Size
A series of genetic search algorithms was performed on the KDEL problem, each allowing a different percentage of the top individuals to contribute genetic material to the next generation (given on the X axis). The other parameters are set as in the third column of Table 2.1. The generation number at which the KDEL sequence was found is plotted on the Y axis.

The second part of this study examined the role of the number of survivors at each generation. Since this value in itself does not alter the computation time, only generation of discovery (not absolute time of discovery) was studied. When 20% of the best individuals are allowed to reproduce at each generation, the solution can be found in 20 generations. When 90% are allowed, the average is 320 generations. Reproduction values of less than 20% tended not to find the solution at all. Too few reproducers lead to premature convergence on local maxima, while too many lead to very slow convergence to the global maximum. Figure 2.6 summarizes this data, and shows that the optimal tradeoff seems to occur at a survival size of about 20%.

The final part of this study looked at the role of mutation. In these experiments, the same problem as above was examined, with varying numbers of mutations in each offspring. Figure 2.7 summarizes the data, which shows that when each offspring is subject to between 1 and 256 mutations, the solution is found on average at the same generation number (around 35). The differences between these values are not significant, showing (surprisingly) that the algorithm's efficiency is tolerant to a wide range of mutation incidences.

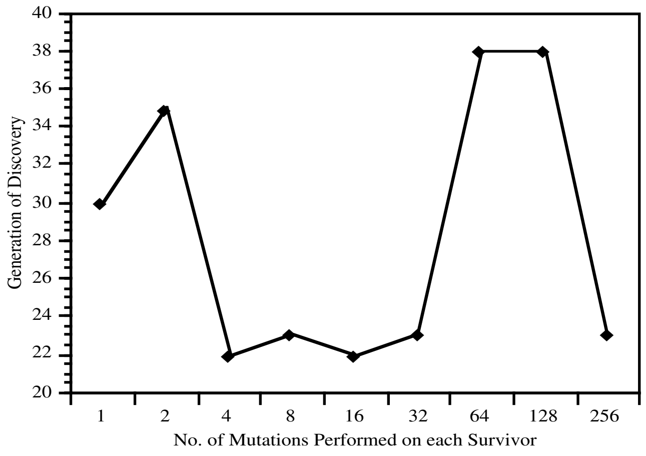

Figure 2.7: Dependence of Generation of Discovery on Mutation Incidence. A series of genetic search algorithms was performed on the KDEL problem, each allowing a different number of mutation events to occur when a top individual contributes genetic material to the next generation (given on the X axis). The other parameters are set as in the third column of Table 2.1. The generation number at which the KDEL sequence was found is plotted on the Y axis.

2.5 Future directions

There are several ways in which this algorithm could be improved. First, it would easily lend itself to parallelization on a computer such as the Connection Machine. Immense savings in time would be accomplished by running the fitness evaluations of each individual in parallel. The algorithm could also be made to deal better with noise in the experimental data by choosing to disregard a member of the "in" group if a schema is found which matches all the other members very well, but does not match it. Other varieties of genetic algorithms (steady-state populations, demes, etc.) may also produce better results.

Acknowledgments

I would like to acknowledge several helpful discussions with David Fogel.

References

Bairoch A., (1991), PROSITE: a dictionary of sites and patterns in proteins, *Nucleic Acids Res.,* 19:2241-2245.

Dandekar, T., (1992), Potential of genetic algorithms in protein folding, *Protein Engineering,* 5(7): 637-645.

Davis, Lawrence, Handbook of Genetic Algorithms, Van Nostrand Reinhold, NY: 1991.

De La Maza, Michael, Tidor, Bruce, (1992), Increased flexibility in genetic algorithms, in *Proceedings of the ORCA CSTS Conference: Computer Science and Operations Research: New Developments in Their Interfaces,* pp. 425-440.

Eldredge, Niles, *Time Frames,* Simon and Schuster, New York: 1985.

Fogel, D. B., (1990), Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems, *Biological Cybernetics,* 63:111-114.

Goldberg, David E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, MA: 1989.

Holland, John H., *Adaptation in Natural and Artificial Systems,* Univ. of Michigan Pr., Ann Arbor: 1975.

Holland, John H., *Adaptation in Natural and Artificial Systems,* MIT Press, MA: 1992.

Koza, John R., *Genetic Programming*, MIT Press, MA: 1992.

Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs,* Springer-Verlag, NY: 1992.

Pelham, H.R.B., (1990), The retention signal for soluble proteins of the endoplasmic reticulum, *Trends Biochem. Sci.,* 15:483-486.

Wells D.E., McBride C., (1989), A comprehensive compilation and alignment of histones and histone genes, *Nucleic Acids Res.,*17:r311-r346.