2008

Applications of genetic algorithms in bioinformatics

Amie Judith Radenbaugh
San Jose State University

APPLICATIONS OF GENETIC ALGORITHMS IN BIOINFORMATICS

A Thesis

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Amie Judith Radenbaugh

May 2008

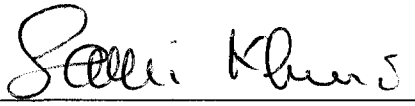
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

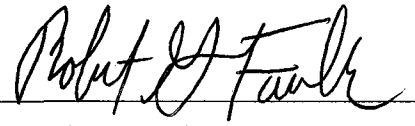
In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

ProQuest LLC
789 E. Eisenhower Parkway
PO Box 1346
Ann Arbor, MI 48106-1346

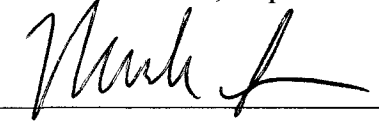
APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE



Dr. Sami Khuri, Department of Computer Science



Dr. Robert Fowler, Department of Biological Sciences



Dr. Mark Stamp, Department of Computer Science

APPROVED FOR THE UNIVERSITY

ABSTRACT

APPLICATIONS OF GENETIC ALGORITHMS IN BIOINFORMATICS

by Amie Judith Radenbaugh

This thesis examines three challenging problems in bioinformatics: Multiple Sequence Alignment, Gene Prediction, and Population Genetics Modeling. It evaluates existing algorithms for the problems and provides implementations of genetic algorithms for each problem. The results from the genetic algorithms are compared to the existing algorithms.

Being able to align multiple sequences of DNA, RNA, or amino acids is essential for biologists to determine similarity in sequences which often leads to similarity in function and provides valuable evolutionary information.

The goal of Gene Prediction is to identify regions of genomic DNA that will encode into proteins. Computational methods are necessary to keep up with the annotation of the rapidly increasing sequencing of genomes.

Modeling population genetics for unpredictable environments provides a tool for improving population forecasts. These forecasts are made by observing past environmental fluctuations on natural selection and how these fluctuations affect population genetics.

ACKNOWLEDGEMENTS

I would like to extend my heartfelt thanks to my advisor, Dr. Sami Khuri, for introducing me to bioinformatics and filling me with enthusiasm for the topics in the field. I have greatly appreciated his guidance and expertise and his constant support and encouragement.

In addition, I would like to thank Dr. Robert Fowler for sharing his biological knowledge and for his participation in our bioinformatics seminar. I thank Dr. Mark Stamp for his early interest in my thesis and for showing me how the genetic algorithms presented here could also be used in information security. Many thanks also to Dr. Chris Brinegar for rekindling my interest in biology through his excellent lectures.

There are many people who contributed one way or another to the completion of this thesis. My thanks to Tom Austin for the outstanding collaboration on the Multiple Sequence Alignment problem. Thanks to Natasha Khuri and Shane Berreen for the brain-storming sessions about the Gene Prediction problem. Thanks to Dr. Ulrich Steiner for the collaboration and for introducing me to Population Genetics. Thanks to the entire Technical Team at TAIR and a big thanks to all of my friends who have stood by me through these past three years.

Finally, I especially would like to thank my parents and Dr. Katrin Erdmann for encouraging me to pursue my Master's Degree and for the countless hours of listening to the ups and downs of the entire process.

Table of Contents

List of Tables

List of Figures

Mendelian Genetics

Gregor Mendel, a Central European monk, spent his early adult life doing basic genetic research. He experimented with selective cross-breading of common pea plants over many generations and noticed that certain traits showed up in offspring without any blending of characteristics. This was extremely important at that time, because the leading biological theory was that inherited traits blend from generation to generation. Mendel determined that genes were passed on to descendents unaltered and that for any particular trait, the parent genes separate and one part of each parent gene is used to form a new gene in the descendent. The part of the parent gene that is passed on to the descendent is a matter of chance (Mendel, 1865).

Mendel (1865) also observed the concept between dominant and recessive genes and that dominant genes do not alter recessive genes in any way so that they can be passed on to successive generations. Assume that traits in parents are represented as two bit strings where 1 represents a dominant part of the gene and 0 represents a recessive part, then the likelihood that a certain trait will be passed on to a child can be easily determined. Obviously, if the mother trait is "11" and the father trait is "11", the child will inherit the "11" trait. If the mother trait is "11" and the father trait is "00", then the child will have a mixed trait with one part "0" and one part "1". The order that they are combined is irrelevant, so that "01" has the same meaning as "10". One could compute the

so called "Punnett Squares" for all possibilities. One of the more interesting possibilities using this simplified schema is:

Table 1. Punnett Square for Offspring Possibilities

	0	1
0	00	10
1	01	11

Source: Author's Research

There is a 25% chance that the child gets either "00" or "11" and a 50% chance that the child gets "01" or "10". The next chapter will introduce genetic algorithms and explain how the ideas behind the discoveries that Mendel made are used to solve computational problems.

Genetic Algorithm Basics

Genetic algorithms use evolutionary techniques to find good approximate solutions. They use survival of the fittest techniques and have self-repair, self-guidance, and reproduction methods. They are highly randomized and are ideal for search and optimization problems.

There are four major steps in a genetic algorithm. The first challenge is to determine how the initial population will be created. Each individual in the population is a possible solution to the problem and should be generated randomly. The biggest challenge in a genetic algorithm is to determine a good fitness function. The fitness function measures the "goodness" of a solution versus other solutions. There is a reproduction phase which is based on both the fitness value and chance. There is a crossover phase where two parent

individuals are chosen and two new children individuals are created by mixing the traits of the two parent individuals. There is a mutation phase where one parent individual is selected and a mutation is done. A mutation on an individual occurs at a very low rate.

The last step in the genetic algorithm is to determine when to terminate. This is typically done in one of two ways. The user can either specify the maximum number of rounds that the genetic algorithm should run, or they can specify a maximum number of rounds to run where the highest fitness value hasn't changed. These are the basics of genetic algorithms. A flow-chart for genetic algorithms is shown in Figure 1:

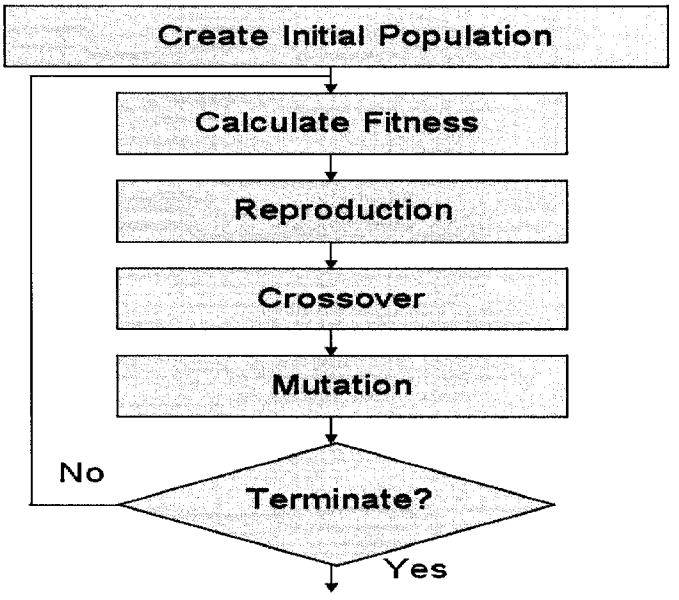


Figure 1. Flow-Chart of a Genetic Algorithm

In every generation, a new set of artificial individuals is created using bits and pieces from the fittest individuals of the population and an occasional new part that is tried for good measure. Although genetic algorithms are randomized,

they efficiently exploit historical information to speculate on new search points with expected improved performance. Features for self-repair, self-guidance, and reproduction are the rule in biological systems, whereas they barely exist in the most sophisticated artificial systems (Goldberg, 1989).

Initial Population

Genetic algorithms require that there is a mapping between the real world representation of a problem and a data structure which can be as simple as a string. The string can be just a sequence of 1s and 0s where parts of the string represent pieces of the real world. In the genetic algorithms that will be introduced in this thesis, the individuals will be alignments of DNA sequences, sets of gene predictions for a DNA sequence, and pairs of alleles making up a trait.

To get started, here is a simple example from Goldberg (1989). For an initial population: assume there is a black-box with five switches on it where the output of the box is a dollar value based upon the configuration of the switches. The goal is to find the combination of the switches being on or off that will lead to the most money coming out of the box. The five switches can be represented as a string of 1s and 0s where 1 means the switch is on and 0 means the switch is off. The population size is four, and the strings are randomly generated by flipping a coin for each 0 or 1 in each string. One of the powerful aspects of genetic algorithms is that they start with a population of strings instead of just one

starting point, which results in the ability to search many solutions in parallel (Goldberg, 1989). A possible initial population is shown in Figure 2:

01101

11000

01000

10011

Figure 2. Initial Population of Strings

Reproduction

Reproduction is the first process in a genetic algorithm where the individual strings in the population are evaluated according to their optimality which is determined by an objective function. Biologists call this the fitness function. The fitness function is just a way to measure the profit, utility, or goodness that is trying to be maximized. Selecting strings according to their fitness values means that strings with a higher fitness value have a higher probability of contributing one or more offspring to the next generation. This correlates to natural selection – survival of the fittest. In natural populations fitness is determined by a creature's ability to survive and reproduce (Goldberg, 1989).

Assume that the fitness function is a simple $f(x) = x^2$. Table 2 shows the initial population from Figure 2, the integer value of the binary string, the fitness values, and the percentage that each of the strings has compared to the total

fitness sum. The simplest way to reproduce is to use the classic "roulette wheel" approach. A roulette wheel is spun and 14.4 % of the time, the ball will land in the slot for the first string, 49.2% of the time, the ball will land in the slot for the second string, etc. This assures that strings with a higher fitness value are most likely to produce offspring for the next generation. The wheel is spun four times, and the new generation is created with the four individuals that were randomly selected from the spins (Goldberg, 1989).

Table 2. Sample Fitness Values

Individual	Integer	Fitness f(x) = x^2	% of Total
01101	13	169	14.4
11000	24	576	49.2
01000	8	64	5.5
10011	19	361	30.9
Total		1170	100

Source: Goldberg, 1989

Crossover

From the new population, pairs of strings are randomly chosen for crossover. Crossover is just a swapping of parts of the strings. A random crossover point is generated, and two new strings are generated by combining the first part of Individual 1 up to the crossover point with the last part of Individual 2 after the crossover point. The other new string will have the first part of Individual 2 up to the crossover point and the last part of Individual 1 after the crossover point. For example, consider the following individuals:

Individual 1 = 0110 | 1

Individual 2 = 1100 | 0

Where the pipe symbol "|" indicates the crossover point. The new individuals after crossover takes place will be:

New Individual 1 = 01100

New Individual 2 = 11001

Mutation

Mutation is the last step in the process. The chance that mutation occurs is very low. For this example, the strings are evaluated on a bit-by-bit basis and the bits are switched from a 0 to a 1 or a 1 to a 0. The probability that this happens at each bit is usually much less than 5%. This also coincides with the feeling for how often mutation occurs in the real world.

Conclusion

The mechanics of genetic algorithms are highly randomized, yet this is one of the main sources of their power. At first it seems surprising that chance should play such a fundamental role, but if some "idea" could be represented as a string, where substrings represent certain "notions" about the idea, then it can be clearly seen that the populations are not just stringent solutions to a problem, but rather "ideas" containing some helpful "notions" and some less helpful "notions" with the goal being to find the optimal "idea". Genetic algorithms keep combining the helpful "notions" from one "idea" with other helpful "notions" from other "ideas" until an optimal "idea" is found (Goldberg, 1989).

The next chapter will expand on some of the basic ideas presented here and show how they apply to the first application of genetic algorithms in bioinformatics that was studied in this thesis.

Multiple Sequence Alignment

Multiple Sequence Alignment (MSA) is a challenging and important problem to be solved in bioinformatics. Being able to align multiple sequences of DNA, RNA, or amino acids is essential for biologists to determine similarity in sequences which often leads to similarity in function and provides valuable evolutionary information. There are a variety of algorithms that exist for finding the most optimal alignment of a given set of sequences, including the Needleman-Wunsch Algorithm, the Smith-Waterman Algorithm, and the use of Hidden Markov Models to name a few.

A literature review on how Genetic Algorithms (GAs) have been used for MSA will be given. The similarities and differences during each phase of the GA will be analyzed. For each phase, either the approach with clearly the best results or the approach that is used by all authors, making it a practical standard, will be considered in the implementation of a GA, called GAMSA (Genetic Algorithm for Multiple Sequence Alignment) (Radenbaugh & Austin, 2006). Most researchers have a common approach on the population representation and on the reproduction phase, but they vary slightly on the crossover and mutation phases and vary highly on the fitness function that should be used.

The intention is to implement a GA for the MSA problem and to find the best approach to take for each phase. The goal is to create a GA that can perform as good as or better than the industry standard ClustalW. ClustalW is designed for global alignments where all of the sequences are aligned over their entire length. The sequences are typically similar and all have roughly the same size. ClustalW can also be used for local alignments where only certain parts of the sequences are aligned, but it is not optimized for local alignments. The sequences for local alignments might be dissimilar and have varying lengths. The goal of this thesis is to investigate how genetic algorithms perform not only on global alignments but also on local alignments.

Review of the Literature

In this section, a review of the literature on how genetic algorithms have been used to solve the MSA problem will be given. Each phase of the genetic algorithm will be reviewed and the similarities and differences will be noted.

Initial Populations

The first challenge of a genetic algorithm is to determine what the individuals of the population will represent and to generate an initial population with some degree of randomness. All of the literature that was reviewed for this thesis (Hernandez, Grass, & Appel, 2004; Horng, Wu, Lin, & Yang, 2005; Shyu, Sheneman, & Foster, 2004; Wang & Lefkowitz, 2005) suggests that each individual in the population should be one multiple alignment of all the given sequences, but the way that they came up with the initial population varies.

Horng et al. (2005), Shyu et al. (2004) and Wang & Lefkowitz (2005) increased the sequence length by a certain percentage and randomly inserted gaps or buffers of gaps into the sequences. Hernandez et al. (2004) took a new approach and used previously developed tools to align the sequences to a certain degree and then used the GA to optimize the alignment.

Another important initial setting is to determine the best population size. There was no consensus for this and all authors mentioned that it is dependent upon the sequence lengths and is best calculated through testing and fine-tuning and should be left as configurable.

Since the approach with inserting random gaps or buffers of gaps to each sequence in the alignment is widely accepted as among the best, it will be used in the implementation of GAMSA. The initial population will be generated in the following way: For each column in the alignment, either the next element in the sequence will be taken or a gap will be inserted. The probability of a gap in a sequence is proportional to its length. This will hopefully achieve a more random dispersal of the elements. To make this clearer, here is an example: suppose that there is an attempt to align these sequences of nucleotides:

```
Sequence 1: ATTGCCGACT

Sequence 2: AC

Sequence 3: GACCCTAG
```

The longest of these sequences is ten nucleotides. The number of gaps to be inserted to every sequence for this example is also ten, so the total

alignment length would be twenty. For each column, there is a 50% chance of inserting a nucleotide for Sequence 1, a 10% chance for Sequence 2, and a 40% chance for Sequence 3. (Note that these are independent probabilities. An element from some, all, or none of these sequences might be inserted). The ending result might look something like this:

```
Sequence 1: ---AT-TGC---C-G-A-CT
Sequence 2: --------AC----------
Sequence 3: --G---AC-C----C-TAG
```

The sequences are now randomly aligned and have enough gaps so that the nucleotides can be shifted later in the algorithm.

Reproduction

All of the authors (Hernandez et al., 2004; Horng et al., 2005; Shyu et al., 2006; Wang & Lefkowitz, 2005) used the typical tournament style, also known as, "roulette wheel" style of reproduction. Two of them (Horng et al., 2006; Wang & Lefkowitz, 2005) also used some sort of elitism while further restrictions were made by Wang & Lefkowitz (2005) to only allow the top scores to reproduce.

As shown with the roulette wheel approach in Section 2.2, for every generation, each alignment in the population is judged according to a fitness function. This will determine the chance of survival for each of these alignments.

After all alignments have been scored, they will be randomly selected using weighted probabilities. The population size will stay the same, but there

may be copies of some of the alignments, and others will disappear. For instance, suppose there are 3 alignments with the following scores:

Alignment 1 = 25

Alignment 2 = 15

Alignment 3 = 10

The total population fitness is the sum of all of the individual alignment fitness scores in the population which is for this example fifty. This means that Alignment 1 will have a 50% chance (Alignment 1 individual fitness divided by the population total fitness) of being selected, Alignment 2 will have a 30% chance, and Alignment 3 will have a 20% chance. A roulette wheel is created with one hundred slots on it. Alignment 1 will occupy slots 1-50, Alignment 2 will occupy slots 51-80, and Alignment 3 will occupy slots 81-100.

A random number will be generated to determine which alignments will be added to the next generation. In the example, three random numbers are generated between 1 and 100. If the numbers are 23, 44, and 92, then the alignments that occupy these slots on the roulette wheel will be added to the next generation. The new population will consist of two copies of Alignment 1 and one copy of Alignment 3. Alignment 2, though not the lowest scoring alignment, dies off.

Finally, since elitism is used by two of the authors and ensures that at least one copy of the highest scoring alignment survives this stage, it will also be used in the implementation of GAMSA.

Crossover

After reproduction, pairs of alignments from the old population are randomly chosen for crossover. The most common type of crossover is called "One Point Crossover" (Hernandez et al., 2004; Shyu et al., 2005) and is the process of dividing the sequences in the alignments at a random point, and then swapping the first halves of the first alignment with the first halves of the second alignment. As an example, assume that the alignments each have three sequences:

Alignment 1:	Alignment 2:
AATTCC	AATTCC
ATC---	A-T-C-
-ATTC-	-AT-TC

A random crossover point for these sequences is generated. For this example, assume the random crossover point is three. All of the sequences in the first alignment will be cut at the crossover point. In order to account for gaps in the sequences in the first alignment, the sequences in the second alignment will be cut according to the number of nucleotides or amino acids in the corresponding sequence in Alignment 1. It is essential to preserve the sequences by not changing the number of nucleotides or amino acids in them. For example, the second sequence in Alignment 1 doesn't have any gaps, but the second sequence in Alignment 2 does, so the second sequence in Alignment 2 will be cut at the fifth position:

```
Alignment 1:          Alignment 2:
AAT | TCC             AAT | TCC
ATC | ---             A-T-C | -
-AT  | TC-            -AT | -TC
```

Now the first parts of all of the sequences from Alignment 1 are swapped with the first parts of all of the sequences from Alignment 2 to generate brand new sequences in each alignment. The result is:

```
Alignment 1:          Alignment 2:
AATTCC                AATTCC
ATC---                A-T-C--
-AT-TC                -ATTC-
```

Another popular form of crossover is called the "Point-to-Point Crossover" (Hernandez et al., 2004). Two random points are generated, and the sequences between these two points are used for crossover. Each nucleotide or amino acid at each position in the crossover range is randomly swapped. A third form of crossover called "Slide-Crossover" is used by Hernandez et al. (2004). Parts of the sequences are shifted left and right to achieve a better alignment. Horng et al. (2005) introduce yet another form of crossover where multiple crossover points are defined and crossover occurs between many blocks.

One Point Crossover is used the most, but the other suggestions by the authors also sound interesting. The GA implementation should definitely support

One Point Crossover, but it might also support other forms of crossover where the type of crossover that is used on each generation is randomly chosen.

Mutation

Mutation is the last step in the process. There are a few ways to do mutation for this problem and they all have to do with gaps and sliding subsequences left or right. Hernandez et al. (2004) have two forms of mutation. They either remove a gap or slide a sub-sequence next to a gap into the gap space which essentially moves the gap from the beginning of the sub-sequence to the end or vice versa. Horng et al. (2005) have four forms of mutation. "MergeSpace" merges two or three spaces together, "MoveSpaceCol" tries to move spaces in the current column to a neighboring column, "FullSpaceCol" adds a column of spaces if the current column has a space, and "MoveRowSpace" selects specific columns and moves spaces in the sequences at these columns to another column. Shyu et al. (2004) randomly select columns in the sequences and then swap nucleotides and spaces in these columns. Wang & Lefkowitz (2005) have three forms of mutation. "Random_gap" randomly inserts a gap into every sequence, or "Local_gap_shuffle" moves one gap in every sequence to a new position in the sequence, and "Block_gap_shuffle" moves multiple gaps in certain columns of the sequences to new positions in the sequence.

Since all of the authors use such a variation at this phase, it is difficult to determine the best course of action. The GA implementation should at least

provide two forms of mutation: one for manipulating gaps and one for sliding subsequences left or right.

Fitness Function

The fitness function determines how "good" an alignment is. The most common strategy that is used by all of the authors, albeit with significant variations, is called the "Sum-Of-Pair" Objective Function. It is typically done in one of two ways:

1) Compare each element in a column to every other element in a column. For nucleotide sequences, use the normal +1 for matches, -1 for mismatches, and -2 for gaps for scoring. For amino acid sequences use the BLOSUM or PAM matrix and the appropriate gap penalty for scoring. The BLOSUM or PAM matrix version should be variable in order to account for different types of sequences (ones that are fairly similar or fairly diverse).
2) For each column, find out what the consensus element is. If there is one, use it as the value to compare all the other values in the columns against. Again, use the typical scoring systems as mentioned in 1.

Hernandez et al. (2004) and Wang & Lefkowitz (2005) create their own scoring matrices based upon the sequences that they are trying to align. Hernandez et al. (2004) measured how unexpected the nucleotide or amino acid frequencies inside the columns are compared to their background frequencies that are estimated from the entire set. The matrix is calculated once at the

beginning and used for the whole GA. Wang & Lefkowitz (2005) creates a library of optimal pair-wise alignments from the sequences that they are trying to align and then evaluates the consistency of the calculated multiple alignment with the ones in the library. Horng et al. (2005) uses the straight-forward Sum-of-Pairs calculation. Shyu et al. (2004) uses the nucleic acid scoring matrix from the International Union of Biochemistry (IUB). This matrix groups nucleotides together according to certain properties, e.g., Purines (A or G) and Pyrimidines (C or T).

Determining the fitness function of the GA is the most difficult part. The function needs to include a biological reference in order to align strands in a biologically relevant way, which is the ultimate goal. The combination of using the PAM and BLOSSUM matrices for amino acids and the IUB matrix for nucleotides could be interesting. Using the consensus sequence as the comparative sequence might also lead to interesting results.

Doping Genetic Algorithms

For most genetic algorithms, the evolution is fairly constant. There is typically a constant percentage probability assigned for each crossover and mutation. Doping genetic algorithms are different. For these, there is a "continuous evolution of the evolution" (Buscema, 2004).

One approach is to use some of the principles of genetic algorithms on the crossovers and mutations themselves. While the actual functions of the crossovers and mutations are not altered in any way, a fitness value is assigned

to them. The fitness value is based upon how much the operation improved the current generation compared to the parent generation. In using this approach, the algorithm can optimize itself while running. This is the approach taken by the Sequence Alignment by Genetic Algorithm (SAGA) package (Notredame & Higgins, 1996).

Unlike most other approaches, SAGA groups mutations and crossovers together and allows them to be selected by their fitness. The strength of combining these two groups is that the balance of crossovers and mutations can shift over the course of the program's execution. However, this also leads to a fundamental problem: crossovers and mutations are not very similar in their effect. Mutations tend to produce stronger results in the short term. Crossovers can break up local maximums and make huge improvements, but in the short term they are often more destructive. As a result, mutations may dominate crossovers without some special care. The solution that the designers of SAGA came up with was to give partial credit to operators for descendents of the children produced by an operator. This seems to resolve the issue.

An alternate approach is used by the GenD algorithm (Buscema, 2004). In this design, the alignments are grouped into tribes. While members of the tribes may interact a little, they are mostly isolated groups. In addition, the algorithm is designed to increase the average health of the population rather than promoting the fitness of the very best. The effect of the tribe approach combined with the promotion of the general health leads to an "inner instability", as the

authors put it. This limited isolation of the population means that they will develop a greater diversity of solutions.

Conclusion

The implementation of GAMSA will use some approaches that are widely accepted as standard and will combine some other successful approaches in an attempt to produce even better multiple sequence alignments. It will have a standard initial representation where random gaps will be inserted and will use the standard tournament style of reproduction with an elitism component. It will provide at least the "One Point Crossover" method. It will have multiple ways of doing mutation which will consist of removing and inserting gaps as well as shifting subsequences left and right. Finally, the fitness function will use the standard "Sum-Of-Pair" scoring method using the PAM and BLOSSUM matrices. Ideally, the IUB matrix will also be included. A comparison between the elements in the sequences to the consensus sequence will be done, which is something that has never been tried before.

GAMSA Implementation

The implementation of the genetic doping algorithm presented in this thesis is called GAMSA. Compared to all of the previously discussed approaches, GAMSA resembles the SAGA design the most. In this section, a description of the approach and comparison to other genetic algorithms will be given, with special emphasis on SAGA.

Basic Process

The main class in the implementation is MultiSeqAligner.java. By default, it is configured to align sequences of amino acids, scoring the alignments using the BLOSUM62 Matrix. A property object can be specified to override these settings.

This class has a findSolution method that takes an array of strings and returns an Alignment object representing the best solution found. The TestCases.java file contains a main method that demonstrates how to use this package. Two amino acid and five DNA sample data sets are provided. The user can enter a number between 0-6 that will correspond to the test that they want to run. To change any default parameters, such as the population size or the number of unchanged rounds needed before GAMSA will terminate, refer to the getDefaultProperties() method in the MutliSeqAligner class.

First, the process will be covered at a high level before going into more detail on key elements of GAMSA's design.

Initial populations. The first step in a genetic algorithm is to generate the initial population. This is only done once each time the program is run.

As with most genetic algorithms, the population is a collection of possible solutions. In the case of GAMSA, the individuals of the population are different alignments of the input sequences. The sequences themselves may be of either nucleotides or of amino acids. The design is flexible enough that alignments of

other sequences might be possible as well, though most likely this would take a certain amount of customization.

By default, the population consists of one hundred alignments. For each sequence in these alignments, gaps are randomly inserted to pad all sequences to the same length. This length is set to 15% of the length of the longest sequence (before padding) in the alignment. These settings can be overridden by specifying "populationSize" and "percentageIncrease" in the configuration options.

Reproduction. For every generation, the first step to be performed is reproduction. Each alignment in the population is given a percentage chance of survival equal to its relative fitness. Every alignment has at least some probability of being selected.

A spot is reserved for the individual(s) with the highest fitness in the population. This individual will be added back to the population after all other steps have been performed, guaranteeing that the best score in the population will never drop in a new generation.

Crossover and mutation. After the new generation has been created, members are selected for crossover or mutation. These may be merged into a single step, but it was found that the best results are achieved by performing these as two distinct steps.

Evaluation of operators. The crossovers and mutations themselves form a population. While they do not change, each operator does have a fitness value.

This is based on the relative increase or decrease in the new alignments they create. If an operator has recently been more successful at creating healthy alignments, it is more likely to be used for future operations.

At this step in the algorithm, each operator's success is determined and its fitness is recalculated.

Fitness Functions

With genetic algorithms, the most critical piece of information is how to score the fitness of a given individual in the population. In GAMSA, each alignment within the population determines its fitness by comparing each pair of its sequences and then summing the scores. However, the logic for scoring the sequences themselves is stored in a separate class called a scorer.

Each scorer implements the gamsa.scorer.Scorer interface and may be specified in the configuration file with the "seqScorer" property. With this design, anyone using GAMSA can create his or her own scoring methods, depending on the nature of the sequences to be aligned.

The two implementations included with GAMSA are DNAScorer and Blosum62Scorer.

DNAScorer. This class is designed to score the alignment of two sequences of nucleotides. Each nucleotide that matches counts for +1. Each pair of nucleotides that is misaligned counts for -1.

Gaps are treated differently. If a gap exists in the middle of one sequence it is scored as -2. Leading or trailing gaps are scored as -1. The main reason for

this is that gaps outside of a sequence should not be overly punished, but at the same time, some penalty is necessary to avoid rewarding sequences that are totally unaligned. Initially, leading and trailing gaps were ignored. This resulted in very bad alignments and lots of out of memory errors.

Blossum62Scorer. This class is used to score sequences of amino acids using the BLOSUM62 matrix. However, gaps in the middle of a sequence are scored differently. At the point where a new gap is started the penalty is -12. Any extension to a new gap by subsequent gaps gets a penalty of -4. Like the DNAScorer, leading and trailing gaps are only punished half the normal amount.

Operators

One of the advantages of GAMSA is that new operators can be easily added. They only need to extend either the gamsa.operator.Mutation class or the gamsa.operator.Crossover class. This allows designers to create their own operators that may be more useful for different types of alignments.

Mutations are excellent for making incremental improvements to individuals in the population. However, they have a noticeable tendency to get stuck in local maximums. Crossovers help to break these up and to mix the best parts of different alignments. While they are less profitable in the short term, they are essential to producing a good solution.

Selection of operators. For doping algorithms, the selection of operators is not fixed. There are multiple ways of doing this, but GAMSA takes a similar approach to SAGA. Like SAGA, the operators are scored based on the success

of the new alignments that they create compared to the parent alignments. However, unlike SAGA, GAMSA does not give any credit for future generations.

The principle reason for SAGA's design is that crossover operations are inherently destructive – splitting an alignment creates new gaps to maintain the alignment of the elements. By crediting crossovers with some of the success of future generations, the designers of SAGA hoped to take care of this situation.

GAMSA takes a simpler approach. The main problem with the SAGA implementation was that the mutations and crossovers were combined into a single pool of operators. Since mutations tend to be more profitable in the short term, they tend to dominate if the operators are mixed together, unless some extra measures are taken.

GAMSA may be configured this way as well (by setting the "mergeOps" parameter to true), but that is not the default. Instead, the operators are divided into separate groups. Since the crossovers are now compared only to other crossovers, they are fairly represented.

No changes operator: No Ops. In most genetic algorithms, there is a (usually fixed) percentage chance of a mutation or crossover. For GAMSA, the entire population goes through both a crossover and a mutation. An operator that makes no changes called a "No op" is used instead. NoopCrossover.java and NoopMutation.java are (respectively) a crossover and a mutation that return the original alignments without modification. Like all operators, the chance of

their selection is determined by their recent success. However, in the case of these operators, their rating is always zero.

This has a rather elegant effect. If crossovers and mutations are producing better and better results, more will happen. If, on the other hand, they are producing poor results, the no ops will dominate. Even though the operators are divided into two groups, the ratio of crossovers and mutations can change in this way.

No ops might also be exploitable to create a new termination condition, though this was not pursued for the current design of GAMSA.

Crossovers. After reproduction, pairs of alignments from the old population, a mother and a father, are randomly chosen for crossover. In order to try to avoid cases where the exact same alignment is chosen for both the mother and the father, the algorithm tries to select new fathers until the alignments are different or until 10 new alignments have been tried. In the rare case that the mother and father are still the exact same alignment, then crossover will have no affect on the alignments.

There are two main forms of Crossover: a "No Operation Crossover" and a "One Point Crossover". As expected, the No Operation Crossover leaves the mother and father alignments untouched. The One Point Crossover divides the sequences in the alignments at a random point, and then swaps the first halves of the first alignment with the first halves of the second alignment. There are

three variations to the One Point Crossover: 1) Gaps at Beginning, 2) Gaps in Middle, 3) Gaps at End.

The One Point Crossover begins by generating a crossover point which is a random number between zero and the length of the sequences in the mother alignment. If the crossover point is zero, then the original mother and father alignments are returned untouched. The process of swapping the first halves of the first alignment with the first halves of the second alignment is not as easy as it might seem. One problem is that the two alignments selected for crossover can have varying lengths. The number of actual nucleotides or amino acids in the alignments is the same, but the alignments can have varying amounts of gaps. For example, assume that the alignments each have the following three sequences where Alignment 1 has sequences of length six and Alignment 2 has sequences of length five:

Alignment 1:	Alignment 2:
A-T-C-	ATCC-
ATC---	--ATC
-ATTC-	AT-TC

The random crossover point is generated and is, for example, at position three. All of the sequences in the first alignment will be cut at the crossover point. In order to account for gaps in the sequences in the first alignment, the sequences in the second alignment will be cut according to the number of nucleotides or amino acids in the corresponding sequence in Alignment 1. It is essential to

preserve the sequences by not changing the number of nucleotides or amino acids in them. For example, the first sequence in Alignment 1 will be cut after "A-T" which contains one gap, but the first sequence in Alignment 2 doesn't have a gap between the A and the T. The first sequence in Alignment 2 will be cut after the second position. The same type of problem occurs in the third sequence of each alignment, but the second sequence shows a different problem. The second sequence of Alignment 1 will be cut after "ATC", but the second sequence of Alignment 2 is padded with some leading gaps and will be cut at the end at position five. The following alignments show where the sequences are cut using the pipe "|" symbol:

```
Alignment 1:          Alignment 2:
A-T | -C-             AT | CC-
ATC | ---             --ATC |
-AT | TC-             AT | -TC
```

Now the first parts of all of the sequences from Alignment 1 are swapped with the first parts of all of the sequences from Alignment 2 to generate brand new sequences in each alignment. The result is:

```
Alignment 1:          Alignment 2:
A-TCC-                AT-C-
ATC                   ---ATC---
-AT-TC                ATTC-
```

Notice now that the sequences within an alignment do not have the same length. The best example is sequence two in each alignment. In Alignment 1, the second sequence only has three elements, while both the first and third sequences have six elements. In Alignment 2, the second sequence has nine elements, while both the first and third have five elements. The sequences within an alignment must have the same length! In order to account for these differences, gaps are added to the sequences.

The three variations of the "One Point Crossover" provide three different ways to add gaps to the sequences. The gaps can be added to the beginning of the sequences, to the middle of the sequences, or to the end of the sequences. The suspicion was that adding the gaps to the middle where the jagged edges from the crossover occurred would prove to be the most effective, but adding them to the beginning and end lead to some interesting results upon crossover in the future generations.

Mutations. Each mutation is designed to make small changes to a single alignment that may improve the design.

BlockShuffleMutation.java takes a single sequence in the input alignment. It selects a random point in the sequence. If it is an element, it looks either to its left or right (again chosen randomly) for the next series of gaps. It will slide that block of elements over until it touches the next block of elements. If a gap is at the chosen position instead, the procedure is the same, except that the block of gaps is moved instead of the block of elements.

GapDeletionMutation.java eliminates a single, randomly chosen gap from each sequence in the alignment. However, if no gap can be found in any one sequence, the operation is aborted, and the original unchanged alignment is returned instead. Interestingly, this tends to be the most profitable mutation in the early stages.

GapInsertionMutation.java is the opposite of GapDeletion. It adds a single gap to each sequence within the alignment. The positions for the gaps are chosen randomly.

GapColumnDeletionMutation deletes all the columns of gaps in an alignment. It loops through each column of every sequence, and if there is a gap in this position in every sequence, then it removes the gap in this column from every sequence.

Termination Conditions

After a population has gone through reproduction, crossover and mutation and a new generation has been formed, it must be evaluated to determine if a solution to the MSA problem has been found or not. If so, the best solution in the population is returned. If not, the population goes through another loop to create yet another new generation. One of the major challenges in designing a genetic algorithm is how to determine when a solution has been found. In general, the goal is to look for some indications that the algorithm has reached a stable point.

GAMSA allows for two different termination conditions to be set. The first is to simply specify the maximum number of rounds that the algorithm will run.

The advantage of this approach is that the running time can be given a maximum bound. The second termination condition is to count the number of rounds that the best fitness score in the population has not changed. Since the best member of the population is preserved each generation, this score will never drop. When it has held steady for a number of generations, it is assumed that the model has reached a stable point. By trial and error, fifty generations seems to be the best setting for this parameter.

Either or both of these conditions can be used in parallel. The parameter "maxRounds" specifies the maximum number of rounds that will be run. If it is omitted, no limit will be used. The parameter "unchangedRoundsNeeded" specifies the number of rounds that the best scoring solution must not be changed for the second condition to be met. If this is a larger value than "maxRounds", it will not be used.

Configuration

GAMSA has a set of default values optimized for amino acid alignment. However, these can be overridden with a configuration file. The constructor for gamsa.MultiSeqAligner takes a property object. Here is a sample file:

```
# Config for calculating proteins, using BLOSUM62
mergeOps=false
seqScorer=gamsa.scorer.Blosum62Scorer
percentageIncrease=15.0
populationSize=100
unchangedRoundsNeeded=50
```

Analysis of Results

The results from GAMSA were compared to the industry standard ClustalW. When run locally, ClustalW was noticeably faster. This was not entirely unexpected. While some care was taken to see that GAMSA was efficient, the emphasis was not on performance. Also, GAMSA is written in Java, and suffers some performance penalty from that.

The results of the tests varied. In one case, GAMSA's answer was different by only a single amino acid! In others, the results were quite different.

In this section some of the results will be covered.

Amino Acid Test Case 1

This test involved seven sequences of amino acids. They varied in both length and similarity. Three different settings for the parameters for the population size (100 and 400) and the number of unchanged rounds needed (20 and 50) were evaluated. The population sizes/unchanged rounds that were tested were 100/20, 100/50, and 400/50. The tests were run on a Pentium 4 machine running Fedora Core 4.

The number of unchanged rounds mostly made a difference in the reliability of the results. While the scores were just as high and the results were just as quick, there were times when the scores would be much worse simply because a local maximum was found.

Population size had a noticeable impact on both the quality and time needed for the results. While the scores ranged roughly from 1000-1500 for a

population of 100, they were 1700-2200 for a population of 400. However, the run times rose from about two minutes to about fifteen minutes.

Amino Acid Test Case 2

For the second test case, ten fairly long sequences of amino acids were used. The sequences were much more similar than those in test case one. Again, population sizes/unchanged rounds of 100/20, 100/50, and 400/50 were used. These tests were performed on the same machine as the previous test.

As with the last case, the unchanged rounds did not make much difference in the score, but seemed to make the results more consistent. However, somewhat surprisingly in this case, the unchanged rounds had a very noticeable effect on performance. On average, the 100/20 case finished in about one third of the time of the other two, and with only a slight decrease in the quality of the results.

Population size seemed to mildly improve the results, but not substantially. Also, its performance increase was not dramatic. In fact, in the examples in the appendix, it took less time than the 100/50 case. One possible explanation for this difference is that these sequences are fairly closely related. As a result, GAMSA seems to make small steps towards the solution, and patience (reflected in the "unchangedRounds" setting) is better rewarded.

DNA Simple Test Case

The first simple sequences got very similar results to those from ClustalW:

```
GAMSA Best Individual:
ATTGCCA-TT
ATGGCCA-TT
ATCCAATTTT
ATCTTC--TT
ATT-------
--GGCCA-T-
ATTG------
Fitness: -74.0

ClustalW Solution Individual:
ATCTTCTT--
ATCCAATTTT
ATT-------
--GGCCAT--
ATGGCCATT-
ATTGCCATT-
------ATTG
Fitness: -84.0
```

For these kinds of short sequences, the results were better when the population size was bigger than the default size of 100. It was interesting to see GAMSA keep the "ATTG" in the last sequence together and either place it at the beginning as shown in this test or at the end of the sequence as in the ClustalW solution.

DNA MYH16 Test Case

This was the best test result received, even outscoring ClustalW. The original sequences were very similar, so it was expected that ClustalW would do an optimal job on aligning them. The solution from GAMSA has only one nucleotide difference to the solution from ClustalW. In the sequence with the gaps, GAMSA created gaps over "CA" and aligned the next "C" to the column

after the gaps. ClustalW created gaps over “AC” and aligned the “C” to the column before the gaps:

```
GAMSA Best Individual:
GAGCAGCTGAACAAGCTGATGACCACCCTCCACAGCACTGCACCCCATTTTGTCCGCTGTATTGTGCCCAATGAGTTTAAGCAGTCAG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTGTCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTGTCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAG--CCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
Fitness: 2322.0
```

```
ClustalW Solution Individual:
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGC--CGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTGTCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTGTCCCCAATGAGTTTAAGCAATCGG
GAGCAGCTGAACAAGCTGATGACCACCCTCCACAGCACTGCACCCCATTTTGTCCGCTGTATTGTGCCCAATGAGTTTAAGCAGTCAG
Fitness: 1728.0
```

DNA Beta Globin Test Case

These original sequences were less similar, and as is the case with ClustalW, GAMSA struggles more with sequences that are less similar. It is interesting to observe the positions where GAMSA starts to insert gaps compared to the positions where ClustalW starts to insert gaps. In this respect, the alignments are quite similar. The entire alignments can be viewed in Appendix B, but shown below are the beginnings of the columns where 4 of the 5 sequences align in the same way as ClustalW:

```
GAMSA Best Individual:
----GTTTACGTTTG
--------ACATTTG
TAGGGCCCCTGCTGC
--------ACACTTG
AGCTAGATTAGTTTC
Fitness: 53.0
```

```
ClustalW Solution Individual:
GTTTACGTTTGCTTC
----ACATTTGCTTC
TAGGGCCCCTGCTGC
----ACACTTGCTTC
AGCTAGATTAGTTTC
Fitness: 352.0
```

DNA HIV Test Case

These sequences are also not very similar and a lot longer than any of the previous tests. The longest sequence length is approximately 650. It is again interesting to compare where gaps are inserted and extended in the GAMSA solution compared to the ClustalW solution. The entire sequences can be found in Appendix B.

DNA BRCA1 Test Case

This test was partially done to evaluate the performance of GAMSA. There were 9 sequences with an approximate length of 800. The population size was increased to 400 and the number of unchanged rounds stayed at 50. The scores converged fairly quickly, but there were small increases at the end leading to the entire run taking over an hour and running close to 2800 rounds. Of course, lowering the population size or the number of unchanged rounds would have taken significantly less time and also given results that are not significantly worse than the final result.

It is again interesting to compare the gaps in the 2 alignments. GAMSA tends to have more gaps on these types of long sequences and does not cluster them together as well as ClustalW. This could be due to the fact that the fitness

function only scores -2 for all gaps and does not distinguish well between gap opening and gap extending as is done for the amino acid sequences. Altering the DNA Scorer to account for this could lead to better results.

Future Research

One of the interesting discoveries was that the "no op" mutations and crossovers gave some indication of the development of the solution. One possibility for a termination condition would be to monitor their comparative fitness to other operators and terminate the program when they become the dominant operation. This might be a better indicator than the current methods. Another interesting approach would be to add in the concept of tribes to GAMSA. By having multiple threads running, with individuals occasionally being copied over to other threads, some of the benefits of the GenD approach (Buscema, 2004) might be achieved. Combining these with the fluctuating operators of SAGA (Notredame et al, 1996) might produce some strong results.

The next chapter will present the second application of genetic algorithms in bioinformatics that was studied in this thesis. The application is very different from the MSA problem and displays the diversity of problems that genetic algorithms can be used to solve.

Gene Prediction

One of the most challenging problems in bioinformatics today is gene prediction. The goal is to identify regions of genomic DNA that will encode into proteins. As more and more genomes are being sequenced, the need to analyze

these data is becoming more and more important. Using traditional experimental methods is costly and time-consuming. The need for accurate computational methods would be very beneficial.

The first step in gene prediction is to locate coding and non-coding regions in a genomic sequence. This can be done using extrinsic homology methods and/or intrinsic computational methods. There are a variety of algorithms and software packages that can be used for gene prediction. The most common and accurate gene prediction approaches will be evaluated.

Genetic algorithms have never been used in the literature for gene prediction in eukaryotes. Genetic algorithms have recently been used for operon prediction in prokaryotes with considerable accuracy (Jacob, Sasikumar, & Nair, 2005). The intention of this project is to implement a genetic algorithm for gene prediction in eukaryotes and to identify the advantages and disadvantages of this approach.

Gene Prediction Software

The general procedure for gene prediction is to obtain a new genomic DNA sequence and to first find the Open Reading Frames (ORFs). For any sequence, there are six ORFs. Typically, the longest ORFs will be the reading frames where the genes are located. There are three basic tests (Mount, 2004) that can be done to verify the ORF:

1) It is known from biology that every third base tends to be the same much more often than by chance. This can be computationally verified.

2) Check the codon frequency. Organisms tend to prefer a specific codon for a specific amino acid. Check to see whether the codons in the ORF correspond to other genes in the organism.
3) Translate the ORF into an amino acid sequence and compare it to a database of existing sequences.

There are several ORF finding software packages: BioEdit is available on the Windows platform. DNA STRIDER is a nice visual tool available for Macintoshes. PLOTORF, GETORF, and SIXPACK are available for the Unix platform. ORF Finder and EMBOSS are available over the web.

The next step in gene prediction is to apply extrinsic content sensors. Extrinsic content sensors are homology based. Approximately 50% of all genes can be found by extrinsic content sensors alone (Mathé, Sagot, Schiex, & Rouzze, 2002). Sequence similarity search methods are used to identify coding and non-coding regions. The genomic DNA is compared to existing DNA or protein sequence databases using tools such as BLAST and CLUSTALW. There are several problems with this approach. If no similar sequences exist, then nothing will be found. It is still hard to determine where the exon and intron boundaries are with these kinds of methods. Small exons are easily missed. The genomic DNA is also compared to existing cDNA or EST databases. cDNA is made up of exons which helps identify the exon and intron boundary. ESTs provide information about partial exons and give hints to alternative splicing, but they only give local and limited information on the gene structure since they are

only partial mRNA (Mathé et al., 2002). Extrinsic content sensors aid in identifying coding vs. non-coding regions, but they don't provide more detailed information about the structure of the gene.

The next step in gene prediction is to apply intrinsic content sensors. Intrinsic content sensors are computational methods. They incorporate knowledge from biology to analyze the data. Determining the probable ORF and verifying the codon composition is done very well with computational methods today. Determining hexamer frequency which is the frequency of 6 nucleotides in a sequence is also done using computational methods. This is used very often with Hidden Markov Models. G+C rich areas tend to have high gene density while A+T areas tend to have low gene density. Therefore, identifying these regions is essential in gene prediction. Lastly, a probabilistic method is applied to the gene prediction problem.

Gene prediction programs use computational approaches such as Hidden Markov Models, Neural Networks, and Bayesian Classifiers. They also focus on different kinds of predictions. They all focus on predicting the exons that will eventually produce a protein, but some software programs also predict promoters, poly-A sites, start and stop codons and others. Lastly, they are trained on data for certain model organisms. Some of the most common model organisms are *Arabidopsis thaliana*, *Caenorhabditis elegans*, *Drosophila melanogaster*, *Mus musculus*, and *Oryza sativa*.

With all of these choices, what is the best gene prediction method? The best method uses a known set of gene structures for training the model. The prediction itself is then done on data that are not in the training set but are similar to it. The gene prediction programs are much better at shorter sequences rather than longer ones. The best approach is to apply multiple methods (Mount, 2004).

After using any gene prediction software, it is essential to verify the predictions that are made. The most advanced gene prediction programs today are only accurate at best 70% of the time.

Literature Review

Since genetic algorithms have not been used for gene prediction in eukaryotes, the review of the literature will focus on general gene prediction algorithms and on how genetic algorithms were used for gene prediction in prokaryotes.

Initial Populations

The first phase of the genetic algorithm is to determine what the individuals in the population will consist of and to generate an initial population with some degree of randomness. The user will provide the genomic DNA sequence. Three methods for generating an initial population will be investigated. The first method needs no prior information about the coding and non-coding regions of the sequence. The second and third methods need prior

information about the coding and non-coding regions that can either be obtained by extrinsic homology methods or by intrinsic computational methods.

The first method is to divide the given sequence into smaller subsequences by randomly selecting a starting point and randomly selecting a length. The length will be dependent on the length of the entire DNA sequence. There could be a parameter for controlling the maximum size of the random length that is generated. For example, the parameter could state that the length cannot be longer than 50% of the entire sequence.

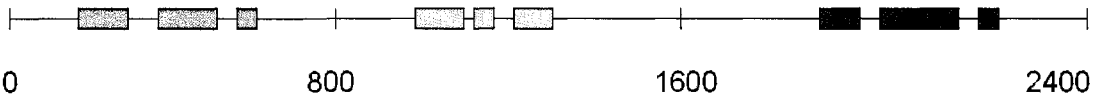


Figure 3. A User Given Genomic DNA Sequence

Consider the user given genomic DNA sequence given in Figure 3 where each box represents an exon and each set of three exons which are color-coded form a separate gene. By randomly selecting a starting point and randomly selecting a length, it is very likely that all of the exons belonging to one gene will not end up in the same subsequence as shown in Figure 4. It is also likely that exons will be cut so that their starting or ending points are not a part of the subsequence as shown in Figure 5. The dependency of the fitness function on having the entire structure of the gene together will be an essential factor.

Start: 1000, Length: 400

Figure 4. Last Exon from Gene 2 Missing

Start: 2000, Length: 200

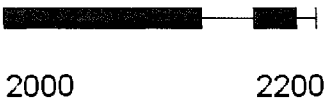


Figure 5. First Exon from Gene 3 Missing, Second Exon Missing Starting Point

A second method would be to randomly choose the number of exons that are allowed in a subsequence. This method assumes that prior information about the coding and non-coding regions of the sequence is known. There are many ways to obtain this information, but the problem that the entire structure of the gene might get destroyed still remains. Consider the user given DNA sequence from Figure 3. The subsequences to be analyzed by the fitness function in the initial population using this method could be:

Start: 1000, Exons: 2

Figure 6. Only Two of the Three Exons from Gene 2 Are Included

Start: 600, Exons: 3

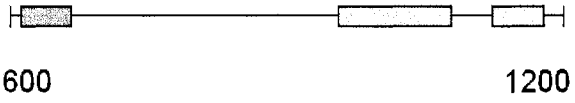


Figure 7. Exon 3 from Gene 1 and Exons 1 and 2 from Gene 2 Are Included

A third method would be to randomly choose the maximum intergenic length between the exons. This method also assumes that prior information about the coding and non-coding regions is known. The minimum and maximum intergenic length could be determined before-hand based on the genes of the organism. This method could be utilized to keep the grouping of the exons together and might aid in speeding up the process by avoiding huge gaps of intergenic material. It will be beneficial to have a minimum and maximum for the random intergenic length. Making the length too small will result in subsequences with no exons and making the length too large will lead to the whole sequence being processed in every individual.

Consider again the user given DNA sequence from Figure 3. Figure 8 shows an individual that could result when the intergenic length is 100. All of the exons from Gene 2 remain intact, because the largest intergenic material is from position 1225 to position 1275 which is less than 100. Figure 9 shows an individual that could result when the intergenic length is 400. All of the exons from Gene 1 and Gene 2 are included in the individual, because the largest intergenic material between the two genes is from position 650 to position 1010 which is less than 400.

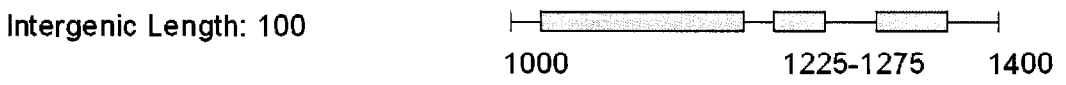


Figure 8. All of the Exons from Gene 2 Remain Intact

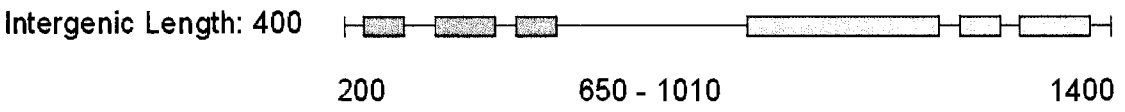


Figure 9. All of the Exons from Gene 1 and Gene 2 Are Included

A similar method to the third method with the random intergenic length was used by Jacob et al. (2005) when doing operon prediction in prokaryotes. The overall accuracy of the prediction algorithm was quite good, but it is difficult to determine if this was one of the main contributing factors. Also, operon prediction in prokaryotes is significantly different from gene prediction in eukaryotes, but it is definitely a method that should be examined further for eukaryotic gene prediction.

Another important initial setting is to determine the best population size. This is likely to be dependent upon the length of the original DNA sequence provided by the user. Therefore, the population size parameter will be left as configurable to the user.

Reproduction

The most well-known type of reproduction is the roulette wheel style of reproduction that was discussed in Sections 2.2 and 4.2. It is likely that some sort of elitism will be used to guarantee the survival of the best solutions.

For every generation, each individual in the population is judged according to the fitness function. This will determine the chance of survival for each of these individuals.

Crossover

After reproduction, pairs of individuals from the current population are randomly chosen for crossover. The most common type of crossover is called "One Point Crossover" which was discussed in Section 3.1.3. Although the representation of the individuals for the MSA problem and this problem are very different, the same concept can be used. "One Point Crossover" is the process of dividing the sequences in the individuals at one random point, and then swapping the first halves of the first individual with the first halves of the second individual. Assume that the following two individuals are randomly selected for crossover:

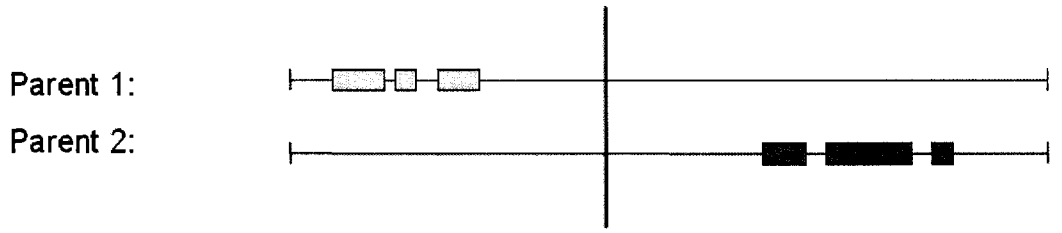


Figure 10. Two Parent Individuals Selected for Crossover

In Parent 1, three exons that belong to the same gene are found. In Parent 2, three exons that belong to the same gene are found where the gene in Parent 2 is different than the gene in Parent 1. Now the random crossover point for these sequences is generated. The random crossover point is indicated by the dark vertical line. The first half of Parent 1 will be combined with the second half of Parent 2. The first half of Parent 2 will be combined with the second half of Parent 1. The result is two new individuals with mixed traits from their

parents:

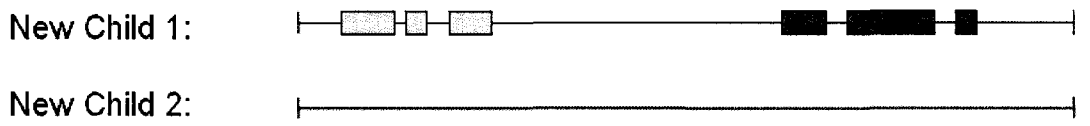


Figure 11. Two New Children Individuals for the Next Generation

The first child will have a very high fitness value and will most likely move on to future generations while the second child will have a very low fitness value and will most likely die out in the next generation.

Mutation

Mutation is the last step in the process. The general idea behind the mutation phase for this problem is to try to repair the beginnings and ends of subsequences that may have been split in the middle of an exon. If a predicted donor or acceptor site doesn't match the expected conserved sequence due to an interruption to the sequence, the starting, or respectively ending, position of the subsequence will be extended in order to include more of the sequence. The hope is that the extension will lead to accurate donor and acceptor sites and ultimately to full exons.

Fitness Function

The fitness function calculates how "good" an individual is. There are many gene prediction programs available today. They use different computational approaches, they focus on certain kinds of predictions, and they

are modelled after specific organisms. This results in a complex matrix when choosing the best gene prediction program for one's specific needs.

In this research, the model organism, *Arabidopsis thaliana*, will be studied. There has already been a lot of research done on *Arabidopsis*, and there are already databases which contain annotation information. This will make it easier to verify the results of the predictions made by the genetic algorithm.

The study by Pavy et al. (1999) showed that GeneMark.hmm was the most accurate gene prediction program for *Arabidopsis*. The original GeneMark program evaluates ORFs and runs in parallel on both DNA strands. It takes into consideration the codon frequency and the G+C vs. A+T content regions. It creates inhomogeneous Markov Models for the coding regions of the DNA sequences from a specific species and homogeneous Markov Models for the non-coding regions of the DNA sequences from a specific species. The algorithm computes and returns an aposteriori probability that a coding region is correct (Borodovsky & McIninch, 1993). This probability score will be used in the genetic algorithm fitness function.

GeneMark.hmm incorporates some of the basic functionality from GeneMark to predict coding and non-coding regions. It then uses a three-periodic Markov Model of order five to exploit hexamer composition. It uses hidden states to represent further characteristics of genes such as the acceptor and donor sites, start and stop codons, and introns themselves (Borodovsky & McIninch, 1993). Unfortunately, GeneMark.hmm does not return any probability

or log-odds score for its predictions. An attempt to integrate GeneMark.hmm is still given and a general approach to incorporating gene prediction software that doesn't return confidence scores is proposed.

Since the last comprehensive study of gene prediction software was done some eight years ago and since GENSCAN is constantly among one of the best gene prediction programs, the fitness function will include predictions from GENSCAN. GENSCAN returns both an overall confidence score for each exon and a log-odds score. GENSCAN and GeneMark.hmm use similar approaches to gene prediction. They both evaluate both strands in parallel, and use species specific information about codon frequency and G+C vs. A+T content regions. GENSCAN also uses three-periodic inhomogeneous fifth order Markov Models for the coding regions of the DNA sequences (Burge & Karlin, 1997). Since GENSCAN and GeneMark.hmm use similar techniques for gene prediction, they will be used together to set the appropriate confidence scores for GeneMark.hmm.

geneid uses a combination of Position Weight Matrices (PWMs) and Markov Models for predicting gene structure. It first scans the sequence from start to finish and uses PWMs to predict acceptor and donor sites and start and stop codons. It then uses these predictions to find the exons of a gene. The exons are scored using the scores from the PWMs in combination with the log-likelihood ratios from a Markov Model for coding regions of DNA sequences from a specific model organism (Parra, Blanco, & Guigó, 2000).

The fitness function of the genetic algorithm will use multiple gene prediction programs that take similar and varying approaches to predict the gene structure of sequences taken from the model organism *Arabidopsis thaliana*. This approach of combining multiple gene prediction programs to work in parallel has never been tried before.

Conclusion

Implementing a genetic algorithm for gene prediction has never been done before. Some of the phases will experiment using new approaches and other phases will use traditional, proven methods. The initial phase is completely experimental, since none of the three approaches have ever been tested. The third method described with the intergenic length has been used in operon prediction for prokaryotes and might be a good approach for eukaryotes as well. The traditional roulette wheel style reproduction with elitism will be used, and the traditional "One Point Crossover" will be implemented. The mutation phase introduces a new approach to assist in repairing interruptions in the sequences. Finally, the possibility of using multiple industry standard gene prediction programs in combination has never been tried before.

Gene Prediction Implementation

The implementation of a genetic algorithm that can be used for gene prediction is presented in this section. A description of the approach that was taken at each phase of the genetic algorithm will be given.

Initialization

The genetic algorithm starts out by reading in the file that contains the user specified DNA sequence. The file should be in the standard FASTA file format. The FASTA format can be used for both DNA sequences and amino acid sequences. The individual nucleotides and amino acids must use their one letter abbreviation e.g. "A" for Adenine, "G" for Guanine, "C" for Cytidine, and "T" for Thymidine for nucleic sequences and "M" for Methionine, "L" for Leucine, "W" for Tryptophan, etc. for amino acid sequences. Each sequence starts with a one line description that starts with the ">" symbol. The description typically contains an identifier and the source of the sequence that can be the database or the genome that it was retrieved from or both. After the one line description, the sequence begins on the next line. Each file may also include multiple sequences. Whenever a ">" symbol is detected, it is assumed that a new sequence is beginning. Figure 12 shows an example of a FASTA formatted file for the AT1G05205 mRNA from *Arabidopsis*:

```
>gi|18390491|ref|NM_100399.1| Arabidopsis thaliana (AT1G05205) mRNA, complete cds
GCGTTTGAGATTTCACCAGGAGCAAGAGAAAGATGAGCGAGACGAGACCAGTGCCGAGGAGAGAGAGTCC
ATGGGGTTTACCGGAAGGTCACCGTGAGCCCAAAGCTCACCGCTGCAACGATCGTGTTGAAGACGTTATC
CAGGCGTTATTCGAGGGAAACCCATTCAAGACAGTTCCAGGACCTTTCAAACTCTTCTACCGATGTATGC
GCTCTAAGCCAGGAGAGGAGCCAACAGAGCCATTCAAATACCTCGACCTGGAACCTCCAAAGAGAGAAGT
TAAACTTGAAGAAGCAAAGCTTGAGTAAAGTGATATTCACTCCCTTTTTTATTCCTGCATACTTTTTAGC
TGAGCGTGTAAGCGCCATCACTGTGATTTTAAGTTTACCTTCAATATGTGTCAACAACATCTTATTTTCT
CTTAATAAAATTGATACTCGGAGTTTC
```

Figure 12. FASTA Formatted File for AT1G05205 from *Arabidopsis thaliana*

The user can configure the directory location and name of the file that should be evaluated. The main class of the algorithm creates the initial population and loops through each phase of the genetic algorithm until a

termination condition is satisfied. The user can define the maximum number of generations or the maximum number of generations where the highest fitness value has not changed or both. The population size is also configurable to the user along with a variety of other parameters that are detailed in Section 4.3.7: Configuration Options.

Initial Populations

After the genetic algorithm has read and parsed the DNA sequence specified by the user, the initial population is created. The individuals in a population represent a possible solution to the problem. In this genetic algorithm, the individuals of the population are subsequences of the initial DNA sequence along with their predicted gene structure.

The initial individuals in the population are created using only a subsequence of the original DNA sequence. Three methods for randomly creating these subsequences were investigated. The first method needs no prior information about the coding and non-coding regions of the sequence. The second and third methods need prior information about the coding and non-coding regions that were obtained using other gene prediction software. The three methods for generating an initial population will be described in this section along with their advantages and disadvantages.

Some gene prediction programs expect to read the sequences from a file and instead of returning the predictions to the standard output, they write the results to a file where the name of the file is the name of the input file with an

extension added to the end (e.g. "lst" is used by GeneMark and GeneMark.hmm). In order to accommodate such gene prediction programs, the original DNA sequence is read in from the user-specified file location and is used to generate a file for each individual in the population. Each individual in the population has to have a separate file, because the sequence for each individual is only a sub-sequence of the original sequence. Each individual is given an index number when it is created. The file that is generated has the index of the individual inserted at the beginning of the file. If a user specifies "AT1G05205.fa" as the name of the file that contains the original DNA sequence to be evaluated. The files that would be generated for the individuals would have the names: "0_AT1G05205.fa", "1_AT1G05205.fa", etc. When the gene prediction programs are executed on the files, they create corresponding output files called "0_AT1G05205.fa.lst", "1_AT1G05205.fa.lst", etc.

Completely random. The original sequence is obtained and for each initial individual in the population, a random starting point and a random length are generated. The individual will be initialized with the subsequence of the original sequence starting at the random starting point and ending after the random length.

The major advantage of this method is that no prior information about coding and non-coding regions is necessary. On the other hand, by randomly chopping up the sequence, the gene structure itself may be destroyed. At first it seemed that this approach to generating the initial population was too

destructive. Gene prediction software relies heavily on the entire sequence remaining intact. In the end, it was found that with a larger population size, all of the individuals with a low fitness were quickly removed from the population, and the completely random approach to generating the initial individuals was sufficient. It is also the preferred approach, because no prior information about the coding and non-coding regions is necessary. The next two methods only allow the sequence to be cut in non-coding regions. Of course, this approach is dependent on the gene prediction software itself that predicts where the coding and non-coding regions are.

Random non-coding regions. The original sequence is obtained and a predefined gene prediction software program is run once on the original sequence. The user can specify which gene prediction software program should be used in the configuration properties. All of the predictions that are returned are marked as coding regions and the parts of the sequence before, after, and between the coding regions are marked as non-coding regions. The sequence can only be cut in a non-coding region. After the non-coding regions are determined, each individual in the population is created in the following way: two non-coding regions are randomly selected, within each non-coding region a random cutting point is generated, the initial individual of the population will contain the subsequence of the original sequence between the two random points.

The advantage to this approach is that the predicted coding regions are kept intact, but it could still be that the individual exons for predicted genes get separated. The major disadvantage of this method is that information about coding and non-coding regions must be known.

Random minimum intergenic length. The original sequence is obtained and a predefined gene prediction software program is run once on the original sequence to again determine coding and non-coding regions. A lower and upper bound for the minimum intergenic length can be specified by the user in the configuration properties file. Within this range, a random number will be generated and used as the minimum intergenic length. All of the non-coding regions where the sequence is shorter than the minimum intergenic length are discarded except for the first and the last non-coding region that are always retained. A cut is only allowed in a non-coding region that is larger than the intergenic length that is specified. This will be done per individual so that there will be a variety of individuals where the minimum intergenic length is possibly different for each one, but they are all within a pre-defined range.

The major advantage of this method is that the individual exons of the genes are likely to be a part of one individual. This allows the genetic algorithm to work on one entire gene at a time. The intergenic length can be specified as appropriate for the model organism that is being worked on. The major disadvantage of this method is again that information about coding and non-coding regions must be known.

Fitness Functions

With genetic algorithms, the most critical part is how to score the fitness of a given individual in the population. The idea behind the fitness function for this implementation was to choose multiple existing gene prediction programs that use similar and different computational approaches for a selected model organism and to combine their results. GeneMark, GeneMark.hmm, GENSCAN, and geneid all use Markov Models to some degree, but they vary the way they are used. They also use other computational methods to find further characteristics of the gene structure ranging from extrinsic homology methods to Position Weight Matrices (PWMs). The tests for this thesis were run on the model organism, *Arabidopsis thaliana.*

Gene prediction programs usually return an overall confidence score or a log-odds score or both. The user can specify the scoring mechanism that they prefer to use in the configuration properties. The results of the tests in this thesis did not vary when using one scoring mechanism versus the other. Unfortunately, some gene prediction software programs do not provide confidence scores in their output. GeneMark.hmm is an example of a program that does not return a confidence score for each predicted exon. An attempt to incorporate GeneMark.hmm was made, and a generic approach to adding such programs has been provided.

The user can configure one or more of the gene prediction software programs to be used in the fitness function. The user can also specify the gene

prediction software program that should be used to set the confidence scores for the programs that don't return a confidence score. If a program does not return a confidence score and the exon that it predicts matches a predicted exon from a program that does return a confidence score, then the confidence score for the program that does not return one is set to the confidence score of the program that does return one. When setting confidence scores for a program, it is recommended that the scores from programs that use similar computational approaches be used.

The fitness function starts by finding all of the predictions for the sequence of the individual from every user specified gene prediction program. It then creates a superset of the predictions. If two prediction programs generate the same exon, then the one with the higher score will be added to the superset. Any predictions made by one program and not by another will be added to the superset.

GENSCAN. GENSCAN is freely available for academic use. There is an academic license agreement form on the GENSCAN web page, and after submitting the form, one can download the appropriate executable. GENSCAN has a parameter file for certain organisms. The parameter file contains information about the characteristics of the organism. It contains information about the gene density, the number of exons per gene, the hexamer composition of coding regions vs. non-coding regions (Burge, 1998) to name a few. The

"Arabidopsis.smat" parameter file was used when running the command-line version of GENSCAN.

The following command was executed from within Java to receive the predictions from GENSCAN:

/usr/bin/genscan /usr/lib/GENSCAN/Arabidopsis.smat <FileName>

Figure 13 shows a sample output file from GENSCAN:

```
Predicted genes/exons:

Gn.Ex Type S .Begin ...End .Len Fr Ph I/Ac Do/T CodRg P.... Tscr..
----- ---- - ------ ------ ---- -- -- ---- ---- ----- ----- ------

 1.01 Init +     33    143  111  2  0   70   17    65 0.795   2.86
 1.02 Intr +    511    590   80  0  2   30   75    68 0.876   2.13
 1.03 Term +    846    930   85  0  1   74   39   102 0.923   5.05
 1.04 PlyA +   1046   1051    6                                1.05
```

Figure 13. GENSCAN Sample Output

The output has thirteen columns that provide detailed information about the predicted genes and or exons. The information that is essential for this genetic algorithm is the start and end indexes located in the fourth and fifth columns labeled "Begin" and "End" and confidence scores. The overall probability that the exon is correctly predicted is in the twelfth column labeled "P...." and the log-odds score is in the thirteenth column labeled "Tscr" for "Total Score". GENSCAN interprets the probability scores as follows (Burge, n.d.):

1) Very high probability GENSCAN exons (e.g., P > 0.99) are almost always exactly correct
2) Moderate- to high-probability GENSCAN exons (e.g., 0.50 < P < 0.99) are exactly correct most of the time, with the likelihood of exact correctness only slightly lower on average than the stated probability.
3) Low-probability GENSCAN exons (P < 0.50) are not reliable and should be treated with caution.

Figure 14 shows a complete explanation of the table (Burge, n.d.):

```
Explanation

Gn.Ex : gene number, exon number (for reference)
Type  : Init = Initial exon (ATG to 5' splice site)
        Intr = Internal exon (3' splice site to 5' splice site)
        Term = Terminal exon (3' splice site to stop codon)
        Sngl = Single-exon gene (ATG to stop)
        Prom = Promoter (TATA box / initation site)
        PlyA = poly-A signal (consensus: AATAAA)
S     : DNA strand (+ = input strand; - = opposite strand)
Begin : beginning of exon or signal (numbered on input strand)
End   : end point of exon or signal (numbered on input strand)
Len   : length of exon or signal (bp)
Fr    : reading frame (a forward strand codon ending at x has frame x mod 3)
Ph    : net phase of exon (exon length modulo 3)
I/Ac  : initiation signal or 3' splice site score (tenth bit units)
Do/T  : 5' splice site or termination signal score (tenth bit units)
CodRg : coding region score (tenth bit units)
P     : probability of exon (sum over all parses containing exon)
Tscr  : exon score (depends on length, I/Ac, Do/T and CodRg scores)
```

Figure 14. Explanation of GENSCAN Output

The GENSCAN executable is run and the predictions are returned to the standard output. The output is parsed and Prediction objects are created for every exon that was returned with a probability score or a log-odds score. Note that Poly-adenine Tails never have a probability score and are thus ignored when

the user specifies that the probability scores should be used. On the other hand, it does return a log-odds score that can be used if this scoring mechanism is selected.

geneid. geneid is freely available under the GNU General Public License. geneid has a parameter file for most model organisms. The parameter file contains information that is used by the Position Weight Matrices (PWMs) for generating acceptor and donor sites and start and stop codons. For example, it includes cut-off points and weights for exons. It also contains transition probabilities for the different characteristics of a gene that are used by the Markov Model. The "arabidopsis.param.Aug_4_2004" parameter file was used when running the command-line version of geneid.

The following command was executed from within Java to receive the predictions from geneid:

/usr/bin/geneid -vP /usr/lib/geneid/arabidopsis_20040804.param <FileName>

Figure 15 shows a sample output file from geneid:

Figure 15. geneid Sample Output

Figure 16 shows the description of each of the geneid columns (Genome Bioinformatics Research Lab, n.d.):

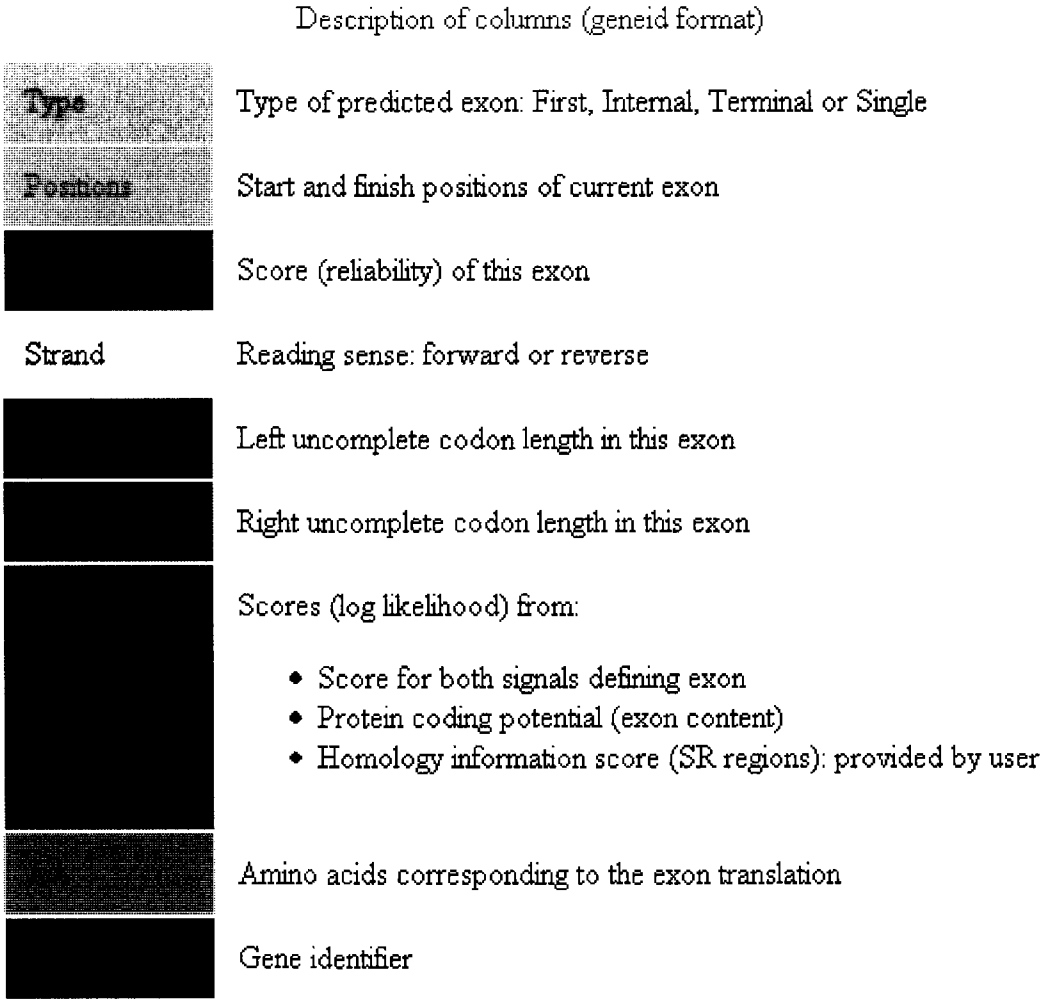


Figure 16. Explanation of geneid Output

As shown in Figures 14 and 15, geneid returns the start and end indexes located in the second and third columns. The log-odds score corresponding to the "Protein coding potential (exon content)" is returned in the eighth column and no overall probability score is returned.

The geneid executable is run and the predictions are returned to the standard output. The output is parsed and Prediction objects are created for every exon that was returned with a log-odds score.

GeneMark and GeneMark.hmm. GeneMark and GeneMark.hmm are also freely available for academic purposes. After faxing in a signed license agreement, an FTP site with a user and password was provided where the program could be downloaded. GeneMark and GeneMark.hmm provide training sets for many model organisms including *Arabidopsis*. The "at_lo_3.mat" and "a_thaliana.mod" training sets were used when executing the command-line version of GeneMark and GeneMark.hmm respectively.

The following command was executed from within Java to receive the predictions from GeneMark:

/usr/bin/gm -lx -m /usr/lib/gm/at_lo_3.mat <FileName>

Figure 17 shows a sample output from GeneMark:

```
List of Open reading frames predicted as CDSs, shown with alternate starts
(regions from start to stop codon w/ coding function >0.50)

Left      Right     DNA         Coding Avg   Start
end       end       Strand      Frame  Prob  Prob
--------  --------  ----------  -----  ----  ----

      33       176  direct      fr 3   0.51  ....
      51       176  direct      fr 3   0.51  ....

List of Regions of interest
(regions from stop to stop codon w/ a signal in between)

   LEnd      REnd    Strand      Frame
 --------  --------  ----------- -----
        6       176  direct      fr 3

List of Protein-Coding Exons
(regions between acceptor and donor site w/ coding function >0.500000)

  Left     Right
  End      End       Strand     Frame   Prob
 -------  -------  ----------- -----  ------
      64      123  direct       fr 3  0.9330
      66       94                     0.9334
```

Figure 17. GeneMark Sample Output

As shown in Figure 16, GeneMark returns a list of "Protein-Coding Exons" with the probability that they are accurate. It also returns the start and end indexes located in the first and second columns labeled "Left End" and "Right End" and the overall confidence score in column four labeled "Prob". Note that GeneMark does not return any log-odds scores, therefore it should only be used in combination with other programs that also return overall confidence scores.

The GeneMark executable is run on the specified file and the program writes the output to a file with the same name with an ".lst" extension added to

the end of it. The output file is then parsed and Prediction objects are created for every exon that was returned with a probability score.

The following command was executed from within Java to receive the predictions from GeneMark.hmm:

/usr/bin/gmhmme3 –m /usr/lib/gmhmme3/a_thaliana.mod <FileName>

Figure 18 shows a sample output file from GeneMark.hmm:

```
Predicted genes/exons

Gene Exon Strand Exon          Exon Range          Exon      Start/End
  #    #         Type                             Length      Frame


  1    1   +   Initial          33        143       111         1 3
  1    2   +   Internal        511        590        80         1 2
  1    3   +   Terminal        846        930        85         3 3
```

Figure 18. GeneMark.hmm Sample Output

As shown in Figure 18, GeneMark.hmm returns the start and end indexes located in the fifth and sixth columns labeled "Exon" and "Range". Notice that GeneMark.hmm does not return any probability or log-odds scores. Since GENSCAN and GeneMark.hmm use similar computational approaches, the scores from GENSCAN can be used to set the scores for GeneMark.hmm.

The GeneMark.hmm executable is run on the specified file and the program writes the output to a file with the same name with an ".lst" extension added to the end of it. The output file is then parsed and Prediction objects are

created for every exon that was returned and prediction scores are assigned from a comparable gene prediction program.

Scoring matrix. The matrix shown in Table 3 summarizes the scores that the gene prediction software provides.

Table 3. Scoring Matrix for Gene Prediction Software Programs

	GENSCAN	geneid	GeneMark	GeneMark.hmm
Probability Scores	Yes	No	Yes	No
Log-odds Scores	Yes	Yes	No	No

Source: Author's Research

Currently the user can choose to use a maximum of three different programs in parallel. If they choose to use probability scores, they can use GENSCAN, GeneMark, and GeneMark.hmm where probability scores for GeneMark.hmm can be assigned from GENSCAN or GeneMark when the predicted exons match. If the user chooses to use log-odds scores, then they can use GENSCAN, geneid, and GeneMark.hmm where the log-odds scores for GeneMark.hmm can be assigned from GENSCAN or geneid when the predicted exons match.

Reproduction

Reproduction is the first step to be performed in the genetic algorithm. Each individual in the population is given a percentage chance of survival equal to its relative fitness. Every individual has at least some probability of being selected. A form of elitism is also used: all individuals with the maximum fitness scored are automatically added to the next generation.

Crossover

After reproduction, two individuals from the population are randomly chosen for crossover. During crossover there are many situations that can occur. There are three cases where individuals are added to the next generation unchanged. Since the individuals are randomly selected, it could be that the same individual is selected twice. In this case, both of the individuals are simply added to the next generation. Crossover is only performed a certain percentage of the time. The user can specify how often crossover should happen as a percentage. Crossover percentages usually range around 70%, but they can vary depending on the type of problem. If two individuals are selected, but the random crossover percentage for these two individuals is greater than the specified crossover percentage, then the two individuals are added to the next generation without any changes. If the two randomly selected individuals don't have any overlapping of the sequence because possibly one individual only covers the beginning of the user given sequence and the other individual only covers the end of the sequence, then the two individuals are added to the next generation untouched.

There are two situations where crossover occurs: when the individuals partially overlap or when one individual is completely contained within another individual. When the individuals only partially overlap, then the randomly generated crossover point has to be within the region where they overlap. When the individuals completely overlap, then the randomly generated crossover point

has to again occur in the region where they overlap which is between the start and end index of the smaller individual. After the random crossover point has been determined, the first and second halves of the individuals are swapped.

Mutations

Mutations occur at a very low rate. The idea here is to randomly add parts of the sequence to the start or end of the sequence. This is especially beneficial if the sequence only contains part of an exon at the beginning or end. By adding part of the sequence to the start or end, the chance that an exon will become whole again is increased.

First, it is randomly decided if the sequence should be extended at the start or the end. If the start is chosen, then a random point is generated from zero to the start of the individual. The algorithm checks to see if the random point is in a coding or non-coding region. It loops until it finds a random point in a non-coding region and then adds all of the sequence from the random point to the original start index to the individual. Similarly, if the end is chosen to be extended, then a random point is generated from the end of the individual to the end of the original sequence. The algorithm again loops until it finds a random point in a non-coding region and then adds all of the sequence from the end of the individual to the random point to the individual. In this way, sequences of the individuals are extended either at the start or end.

Termination Conditions

After a population has gone through reproduction, crossover and mutation and a new generation has been formed, the new generation is evaluated to see if the algorithm should halt. The genetic algorithm finds the individual with the maximum fitness. If the fitness is greater than the maximum fitness thus far, then it is recorded and the algorithm continues. If the maximum fitness is the same as the maximum fitness thus far, then the count of the number of generations with this maximum fitness is incremented. If the number of generations with this maximum fitness is equal to the "UnchangedRoundsNeeded" parameter that the user may specify, then the algorithm terminates. If the parameter is not specified, then the algorithm is run as many times as specified by the "GenerationSize" parameter. These parameters can be used separately or in combination. The goal is to allow the algorithm to reach some kind of stable point.

Configuration Options

The user is allowed to specify a variety of parameters to influence the behavior of the genetic algorithm. The constructor for predictor.PredictorMain takes a Property object as a parameter. Here is a sample file:

```
InitialSequencePath="/home/user/workspace/GenePrediction/co
de/cfg/sequences/";
InitialSequenceFileName="AT1G05205_1.txt";
PopulationSize="10";
GenerationSize="5";
UnchangedRoundsNeeded="5";
ProbabilityCrossover="75";
ProbabilityMutation="5";
```

```
GenscanExecutablePath="/usr/bin/genscan";
GenscanModelPath="/usr/lib/GENSCAN/Arabidopsis.smat";
GeneMarkExecutablePath="/usr/bin/gm";
GeneMarkModel="/usr/lib/gm/at_lo_3.mat";
GeneMarkHmmExecutablePath="/usr/bin/gmhmme3";
GeneMarkHmmModel="/usr/lib/gmhmme3/a_thaliana.mod";
GeneIdExecutablePath="/usr/bin/geneid";
GeneIdModel="/usr/lib/geneid/arabidopsis_20040804.param";

// 0=Completely Random, 1=Non-Coding Regions, 2=Intergenic
Length
InitialPopulationType="0";

// 0=Genscan, 1=GeneMark, 2=GeneId
NonCodingGenePredictionSoftware="0";
IntergenicLengthLowerBound="50";
IntergenicLengthUpperBound="100";

FitnessUseGenscan="0";
FitnessUseGeneMarkHmm="0";
FitnessUseGeneMark="0";
FitnessUseGeneId="1";
```

Analysis of Results

In this section two tests will be evaluated. The actual resulting predictions are only as good as the gene prediction programs themselves, but the results will demonstrate the advantage of using multiple gene prediction programs in parallel.

Single Gene Test

This test was done on a small DNA sequence called "AT1G05205.1" from *Arabidopsis thaliana*. The DNA sequence itself was shown in Figure 12 as an example of a FASTA formatted file. The gene has three exons, two introns, and a 5' Untranslated Region (UTR) and a 3' UTR. Figure 19 shows the GBrowse

visualization of the gene from The Arabidopsis Information Resource (TAIR) (n.d.):

Figure 19. GBrowse Visualization of AT1G05205.1

The dark blue boxes represent the coding regions, the lines between the coding regions are the introns, and the 5' and 3' UTR regions are represented by the light blue boxes. Figure 20 shows a summary of the gene features from the TAIR website:

Gene Feature	type	coordinates	annotation source	date
	ORF	33-930		
	5' utr	1-32		
	coding_region	33-143		
	coding_region	511-590		
	coding_region	846-930		
	exon	1-143		
	intron	144-510		
	exon	511-590		
	intron	591-845		
	exon	846-1069		
	3' utr	931-1069		

Figure 20. Gene Features for AT1G05205.1

The different characteristics of the gene are specified. The labeled exons in Figure 20 include the 5' and 3' UTR regions. Most gene prediction programs do not include the 5' and 3' UTR regions in their predictions. The exons that they predict correspond to the coding regions that are listed above which are the

regions that actually encode for a protein. The coordinates for the coding regions are Exon 1: 33-143, Exon 2: 511-590, Exon 3: 846-930.

The genetic algorithm was run on this sequence with the following parameters:

```
PopulationSize="200";
GenerationSize="10";
UnchangedRoundsNeeded="3";
ProbabilityCrossover="75";
ProbabilityMutation="5";
```

The log-odds scores from GENSCAN and geneid were used in parallel and the initial population was created using the "Completely Random" method. Figure 21 shows the predictions obtained from GENSCAN:

Gn.Ex	Type	S	.Begin	...End	.Len	Fr	Ph	I/Ac	Do/T	CodRg	P....	Tscr..
1.01	Init	+	33	143	111	2	0	70	17	65	0.795	2.86
1.02	Intr	+	511	590	80	0	2	30	75	68	0.876	2.13
1.03	Term	+	846	930	85	0	1	74	39	102	0.923	5.05
1.04	PlyA	+	1046	1051	6							1.05

Figure 21. GENSCAN Predictions for AT1G05205.1

Notice that GENSCAN makes all of the correct predictions and returns the log-odds scores for the predictions. Figure 22 shows the predictions obtained from geneid:

```
Internal   20    143   3.82   + 1 0   1.47   1.23   17.33   0.00 AA  1:42 >AT1G05205.1_1
Internal   511   590   2.49   + 0 2   5.42   2.29   4.22    0.00 AA 43:69 >AT1G05205.1_1
Terminal   846   930   3.23   + 1 0   4.46   0.00   12.38   0.00 AA 69:97 >AT1G05205.1_1
```

Figure 22. geneid Predictions for AT1G05205.1

Notice that geneid correctly predicts the second and third exons but the start coordinate for the first exon is incorrect. Also notice the log-odds scores for the

exons in the eighth column. The second exon from geneid has a log-odds score of 5.42 while the second exon from GENSCAN has a log-odds score of 2.13. All of the other log-odds scores are higher from GENSCAN. Since the superset of exons with the highest scores is used, the resulting solution from the genetic algorithm will contain the second exon from geneid. Figure 23 shows an individual with the maximum fitness from the final population from the genetic algorithm that displays this result:

```
1.01    Init      +     33      143     111     0 0     81.6    3.06
        Internal  +     511     590             0 2             2.49
1.03    Term      +     846     930     85      1 1     92.7    5.05
1.04    PlyA      +     1046    1051    6                       1.05
Indexes(29,1068) Fitness: 1458
```

Figure 23. Genetic Algorithm Predictions for AT1G05205.1

The "fittest" individual had a start index of 29 and an end index of 1068, and all of the exons that were predicted came from the GENSCAN predictions except for the second exon which came from the geneid predictions as expected.

With small sequences, all of the gene prediction programs perform quite well. Even though the most destructive method for creating the initial population was used, the genetic algorithm stabilized only after five generations. With a population size of 200, a generation size of ten, and the number of unchanged rounds at three, the genetic algorithm stopped after just five generations and already had 15 individuals with the maximum fitness. The start and end indexes of the sequences varied slightly, but every individual started before the first exon and ended after the last exon and made all of the correct exon predictions.

Multiple Gene Test

Another test was done on the first 10,000 nucleotides from Chromosome 1 from *Arabidopsis thaliana*. There are two genes in the first 10,000 nucleotides where one of the genes has two variations due to alternative splicing. The first gene called "AT1G01010.1" has six exons, five introns, and the 5' and 3' UTRs. The second gene called "AT1G01020" has two variations "AT1G01020.1" and "AT1G01020.2". The second gene is also located on the reverse strand of the DNA which can be seen by the direction of the arrows in Figure 22. This means that the 5' UTR is located on the right end and the 3' UTR is located on the left end. "AT1G01020.1" has nine exons, eight introns, and the 5' and 3' UTRs. "AT1G01020.2" has seven exons, seven introns, and the 5' and 3' UTRs where the 3' UTR starts before the last intron and continues to the end. Figure 24 shows the GBrowse visualization of the first 10,000 nucleotides from Chromosome 1 from TAIR:

Figure 24. GBrowse Visualization of Chromosome 1 from 1 to 10,000

Figure 25 shows a summary of the gene features for "AT1G01010.1" from the TAIR website:

Gene Feature	type	coordinates	annotation source	date
	ORF	130-2000		
	5' utr	1-129		
	coding_region	130-283		
	coding_region	366-646		
	coding_region	856-975		
	coding_region	1076-1465		
	coding_region	1544-1696		
	coding_region	1809-2000		
	exon	1-283		
	intron	284-365		
	exon	366-646		
	intron	647-855		
	exon	856-975		
	intron	976-1075		
	exon	1076-1465		
	intron	1466-1543		
	exon	1544-1696		
	intron	1697-1808		
	exon	1809-2269		
	3' utr	2001-2269		

Figure 25. Gene Features for AT1G01010.1

Figure 26 shows a summary of the gene features for "AT1G01020.1" from the TAIR website:

Gene Feature	type	coordinates	annotation source	date
	ORF	72-1823		
	5' utr	1-71		
	coding_region	72-167		
	coding_region	274-321		
	coding_region	413-502		
	coding_region	751-796		
	coding_region	903-976		
	coding_region	1089-1174		
	coding_region	1288-1354		
	coding_region	1506-1581		
	coding_region	1669-1823		
	exon	1-167		
	intron	168-273		
	exon	274-321		
	intron	322-412		
	exon	413-502		
	intron	503-750		
	exon	751-796		
	intron	797-902		
	exon	903-976		
	intron	977-1088		
	exon	1089-1174		
	intron	1175-1287		
	exon	1288-1354		
	intron	1355-1505		
	exon	1506-1581		
	intron	1582-1668		
	exon	1669-1948		
	3' utr	1824-1948		

Figure 26. Gene Features for AT1G01020.1

Figure 27 shows a summary of the gene features for "AT1G01020.2" from the TAIR website:

Gene Feature	type	coordinates	annotation source	date
	ORF	72-1423		
	5' utr	1-71		
	coding_region	72-167		
	coding_region	274-321		
	coding_region	413-502		
	coding_region	751-796		
	coding_region	903-976		
	coding_region	1089-1174		
	coding_region	1288-1423		
	exon	1-167		
	intron	168-273		
	exon	274-321		
	intron	322-412		
	exon	413-502		
	intron	503-750		
	exon	751-796		
	intron	797-902		
	exon	903-976		
	intron	977-1088		
	exon	1089-1174		
	intron	1175-1287		
	exon	1288-1581		
	intron	1582-1668		
	exon	1669-1948		
	3' utr	1424-1581		
	3' utr	1669-1948		

Figure 27. Gene Features for AT1G01020.2

Again, only the coding regions that are shown for the genes are of interest. The genetic algorithm was run on this sequence with the following parameters:

```
PopulationSize="50";
GenerationSize="10";
UnchangedRoundsNeeded="3";
ProbabilityCrossover="75";
ProbabilityMutation="5";
```

The log-odds scores from GENSCAN and geneid were used in parallel and the initial population was created using the "Completely Random" method. Figure 28 shows the predictions obtained from GENSCAN:

Gn.Ex	Type	S	.Begin	...End	.Len	Fr	Ph	I/Ac	Do/T	CodRg	P....	Tscr..
1.01	Init	+	264	327	64	2	1	51	82	27	0.710	4.76
1.02	Intr	+	3641	3913	273	0	0	59	87	352	0.991	33.79
1.03	Intr	+	3996	4276	281	1	2	43	5	225	0.954	10.97
1.04	Intr	+	4486	4605	120	0	0	39	89	113	0.997	11.27
1.05	Intr	+	4706	5095	390	1	0	83	103	455	0.999	45.49
1.06	Intr	+	5174	5326	153	1	0	74	9	191	0.999	14.15
1.07	Term	+	5439	5630	192	2	0	92	48	152	0.994	13.04
1.08	PlyA	+	5872	5877	6							1.05
2.06	PlyA	-	6396	6391	6							-3.84
2.05	Term	-	6594	6428	167	1	2	-62	36	190	0.139	0.90
2.04	Intr	-	7232	7157	76	2	1	66	77	2	0.178	0.07
2.03	Intr	-	8325	8236	90	0	0	70	98	30	0.966	6.47
2.02	Intr	-	8464	8417	48	1	0	36	26	117	0.897	3.26
2.01	Init	-	8666	8625	42	2	0	86	65	20	0.802	4.97

Figure 28. GENSCAN Predictions for Chromosome 1 from 1 to 10,000

GENSCAN falsely predicts exon 1.01. The start of exon 1.02 is predicted at 3641 but the real start is at 3761. The end of exon 1.02 is correctly predicted. The rest of the exons for gene one are correct.

GENSCAN predicts the start of exon 2.01 correctly, but not the end. It correctly predicts exons 2.02 and 2.03, but then it misses four exons. Exon 2.04 is also correct, but the last exon is predicted at 6428-6594 when it really is at 6914-7068.

Figure 29 shows the predictions obtained from geneid:

```
# Gene 1 (Forward). 6 exons. 455 aa. Score = 30.36
Internal   3686   3913   5.02   + 2 1   -0.75   2.14   23.18   0.00 AA   1: 77 Chr1:1..10000_1
Internal   3996   4276   8.21   + 2 0    5.58   1.81   21.19   0.00 AA  77:170 Chr1:1..10000_1
Internal   4486   4605   1.20   + 0 0    0.22   2.20   10.37   0.00 AA 171:210 Chr1:1..10000_1
Internal   4706   5095   9.17   + 0 0    4.36   4.64   20.90   0.00 AA 211:340 Chr1:1..10000_1
Internal   5174   5326   3.69   + 0 0    3.91   0.68   13.45   0.00 AA 341:391 Chr1:1..10000_1
Terminal   5439   5630   3.06   + 0 0    3.46   0.00   13.75   0.00 AA 392:455 Chr1:1..10000_1

# Gene 2 (Reverse). 1 exons. 76 aa. Score = 6.03
  Single   6428   6655   6.03   - 0 0    0.00   -1.95  32.27   0.00 AA   1: 76 Chr1:1..10000_2

# Gene 3 (Reverse). 5 exons. 97 aa. Score = 4.51
Terminal   7208   7232  -1.30   - 1 0    0.00   5.13   -1.82   0.00 AA  89: 97 Chr1:1..10000_3
Internal   7564   7649   0.71   - 0 2    3.11   2.98    2.17   0.00 AA  61: 89 Chr1:1..10000_3
Internal   8236   8325   1.59   - 0 0    4.51   4.31   -0.40   0.00 AA  31: 60 Chr1:1..10000_3
Internal   8417   8464   2.44   - 0 0    1.50   3.94    8.30   0.00 AA  15: 30 Chr1:1..10000_3
   First   8625   8666   1.06   - 0 0    2.25   3.56    3.65   0.00 AA   1: 14 Chr1:1..10000_3
```

Figure 29. geneid Predictions for Chromosome 1 from 1 to 10,000

The start of the first exon is predicted at 3686 but the real start is at 3761. The end of the first exon is correctly predicted. The rest of the exons for gene 1 are correct.

For the second gene, geneid makes a false single exon prediction under "Gene 2". Under "Gene 3", the first three exons are correctly predicted. At this point, geneid finds an exon that GENSCAN missed. The last exon has the correct end position, but the start is 7208 instead of 7157.

Figure 39 shows an individual with the maximum fitness from the final population from the genetic algorithm:

```
1.01    Init      +     264     327     64      0 1     91.10   4.76
1.02    Intr      +     3641    3913    273     1 0     98.9    33.79
1.03    Intr      +     3996    4276    281     2 2     95.39   10.97
1.04    Intr      +     4486    4605    120     1 0     99.7    11.27
1.05    Intr      +     4706    5095    390     2 0     99.9    45.49
1.06    Intr      +     5174    5326    153     2 0     99.9    14.15
1.07    Term      +     5439    5630    192     0 0     99.4    13.04
1.08    PlyA      +     5872    5877    6                       1.05
2.06    PlyA      -     6391    6396    6                       -3.84
2.05    Term      -     6428    6594    167     2 2     13.90   0.90
2.04    Intr      -     7157    7232    76      0 1     17.9    0.07
        Internal  -     7564    7649            0 2             3.11
2.03    Intr      -     8236    8325    90      1 0     96.7    6.47
2.02    Intr      -     8417    8464    48      2 0     89.8    3.26
2.01    Init      -     8625    8666    42      0 0     80.30   4.97
Indexes(233,9528) Fitness: 15330
```

Figure 30. Genetic Algorithm Predictions for AT1G05205.1

The genetic algorithm returns the superset of the predictions from GENSCAN and geneid. For the first gene, it returns the first falsely predicted exon from GENSCAN. All of the scores from GENSCAN are higher than those returned from geneid, so all of the GENSCAN predictions are returned.

For the second gene, again all of the scores from GENSCAN are higher than those returned from geneid, so all of the GENSCAN predictions are returned. Here, the true benefit of being able to combine the results from multiple gene prediction programs can be seen. The correctly predicted fourth exon from geneid was missed by GENSCAN, and the genetic algorithm returns it as part of the final solution.

Conclusion

Genetic algorithms provide a flexible platform for integrating multiple gene prediction programs and using them in parallel. Gene prediction programs use

various computational approaches and by combining the results, better predictions can be made.

As gene prediction programs continue to improve, the necessity for defined input and output interfaces is becoming more popular. geneid now uses Gene Finding Format (GFF) which is a standard format for describing gene structure for its output. As more and more gene prediction programs standardize their input and output, it will be easier to integrate them into a hybrid system.

The next chapter will present the third application of genetic algorithms in bioinformatics that was studied in this thesis. Unlike the first and second applications, this application is not aimed at "solving" a problem. Rather the focus is in on modeling various environments and how they change over time.

Population Genetics

Population Genetics is the study of allele frequency distribution and fluctuation due to the four factors that define evolution: natural selection, genetic drift, mutation, and gene flow. In order to understand some of the concepts behind population genetics, a brief review of some basic biological principles is given.

A character is some kind of attribute that allows one organism to be compared to another. For example, the color of the petals of a flower is a character. A trait is a distinct phenotypic character of an organism that may be inherited, environmentally influenced, or a combination of the two. When considering the color of the petals of a flower character, the traits would be the

actual colors that can be seen such as red, orange, or yellow. A single gene controls the color of the petals, even though the alleles that make up the gene can be different. One allele-pair might result in red petals while another might result in yellow petals. The resulting color depends on the two alleles that the gene has and how they interact with each other.

The genotype of an individual describes the genetic make-up, the DNA, of the individual. Diploid organisms like human beings, contain two copies of each chromosome – one from the mother and one from the father. The genes contain one allele from one chromosome and one allele from the other. Alleles can also be dominant or recessive. For example, there are 3 possible genotypes for some gene that has two alleles: dominant R and recessive r. The 3 possible genotypes are RR (homozygous dominant), Rr (heterozygous), and rr (homozygous recessive).

The phenotype is any observable trait of an individual. For the three genotypes listed above, let R be the dominant allele for red flower petals, then both genotypes RR and Rr will produce the phenotype for red flower petals. Some phenotypes are controlled entirely by the individual's genes. Others are controlled by genes but are also influenced by extra-genetic or environmental factors.

The interaction between genotypes and phenotypes can be thought of as the genotype plus environmental factors which will give you the phenotype. The genotype is known by looking at the DNA, while the phenotype is known by

observing an outward appearance of an organism. In 1974, Lewontin proposed the goal of mapping "genotypic space" and "phenotypic space" as follows:

> The challenge of a *complete* theory of population genetics is to provide a set of laws that predictably map a population of genotypes (*G*1) to a phenotype space (*P*1), where selection takes place, and another set of laws that map the resulting population (*P*2) back to genotype space (*G*2) where Mendelian genetics can predict the next generation of genotypes, thus completing the cycle.

$$G_1 \rightarrow^{T_1} P_1 \rightarrow^{T_2} P_2 \rightarrow^{T_3} G_2 \rightarrow^{T_4} G_1' \rightarrow \cdots$$

The goal is, of course, gigantic, but small steps are being taken every day.

Natural selection is the process where traits that are favorable in individuals are inherited and increase in successive generations while traits that are unfavorable will decrease. Genetic drift is the tendency of an allele distribution in a population with a limited size to statistically alter over time due to random events. The fluctuation of the distribution may cause an allele and the traits that are made up of the allele to increase or decrease over successive generations.

The goal of this project was to be able to model population genetics for unpredictable environments. The underlying question is: can an improvement to population forecasts be made based on past environmental fluctuations on natural selection and their affects on population genetics? The first goal of this project was to model a stable environment where the selection on the alleles

remains constant. The second goal was to model an environment where the selection on one or more alleles fluctuates over time while the other alleles remain constant. The third goal was to model the hitch-hiking effect. If the selection on one or more alleles fluctuates over time and is not paired with certain other alleles, then the prediction is that the alleles that do not pair with the fluctuating allele will increase over time while the alleles that do pair with the fluctuating allele will decrease over time.

Population Genetics Implementation

The implementation of a genetic algorithm that can be used for modeling different environments in population genetics is presented. In this section, a description of the algorithm will be given.

Initialization

The genetic algorithm starts out by reading in the configuration properties that the user specifies for the situation that they want to model. One of the most important parameters is the number of alleles that they want to model. The user can also define the standard genetic algorithm parameters such as the population size and the maximum number of generations or the maximum number of generations where the highest fitness value has not changed or both. The complete list of parameters that is available is detailed in Section 5.1.8: Configuration Options.

The main class of the algorithm starts out by creating the initial population and looping through each phase of the genetic algorithm until a termination condition is satisfied.

Initial Populations

After the genetic algorithm has read in the user specified parameters, the initial population is created. The user specifies the number of alleles that they want to model. The individuals in the population are diploid organisms that contain two copies of each chromosome – one from the mother and one from the father. The individuals represent traits or, for diploid organisms, pairs of alleles in the population. It is assumed that all combinations of allele pairs are possible unless the user specifies that certain pairs are not possible in one of the configuration parameters. For each population, the frequency of the individuals or traits is monitored along with the frequency of the single alleles themselves.

The user can specify that certain combinations of alleles are not possible with the "Nonpairs" parameter. The user can also specify a comma separated list of allele pairs that are not allowed to combine for a trait. The allele pairs themselves are separated by a hyphen. If allele "0" cannot pair with either allele "1" or allele "2", then the parameter would be: "Nonpairs" = "0-1, 0-2";

There are two methods for randomly creating the initial population. The first method assumes that the frequency of each allele is initially equal. The second method creates a random distribution of the alleles so that the initial frequencies of the alleles are unequal. The first method is useful for initially

testing the situation that one wants to model. The second method is more realistic to what is seen in nature where the frequencies of alleles will differ.

Equal allele frequencies. The user specifies the number of alleles that they want to model. Each individual will be randomly assigned two alleles – one from the mother and one from the father. The overall distribution of the allele-pairs will be equal and the overall distribution of alleles will be equal.

If the user specifies that the number of alleles that they want to model is four, then the genetic algorithm will create the following alleles: {0, 1, 2, 3}. There are 4^2 possible allele pair combinations. They are: {00, 01, 02, 03, 10, 11, 12, 13, 20, 21, 22, 23, 30, 31, 32, 33}. An example of an initial population with equal frequencies could have the distribution of traits shown in Figure 31:

```
Trait Frequencies:
[00]=6.0
[01]=6.5
[02]=6.270000457763672
[03]=6.109999656677246
[10]=6.899999618530273
[11]=6.549999713897705
[12]=6.079999923706055
[13]=6.109999656677246
[20]=6.260000228881836
[21]=6.380000114440918
[22]=6.329999923706055
[23]=6.139999866485596
[30]=6.050000190734863
[31]=5.980000019073486
[32]=6.420000076293945
[33]=5.920000076293945
```

Figure 31. Trait Frequencies for an Initial Population with Equal Frequencies

And the distribution of alleles shown in Figure 32:

```
Allele Frequencies:
[0]=25.044998168945312
[1]=25.525001525878906
[2]=25.104999542236328
[3]=24.32499885559082
```

Figure 32. Allele Frequencies for an Initial Population with Equal Frequencies

Unequal allele frequencies. The user again specifies the number of alleles that they want to model. For each allele, a random percentage is generated which corresponds to the percentage of the population that will be created with this allele in the trait. The allele will be randomly set as the first or second allele in the trait. The other allele in the trait will be randomly selected from all possible alleles. The distribution of both traits and alleles will therefore be unequal.

If the user again specifies that the number of alleles that they want to model is four, then a possible random percentage of the population that will be created with this allele in the trait could be:

```
InitialFrequency[0]=17
InitialFrequency[1]=63
InitialFrequency[2]=13
InitialFrequency[3]=7
```

Figure 33. Random Percentages of Population with Given Allele in Trait

The overall trait and allele frequencies will be slightly higher or lower than the initial frequencies because at least one allele per individual is still randomly selected from all possible alleles. A possible population with the initial

frequencies from Figure 33 could consist of the following trait and allele frequencies:

```
Trait Frequencies:
[00]=1.0
[01]=12.0
[02]=4.0
[03]=3.0
[10]=6.0
[11]=14.0
[12]=13.0
[13]=6.0
[20]=6.0
[21]=9.0
[22]=1.0
[23]=4.0
[30]=5.0
[31]=10.0
[32]=4.0
[33]=2.0

Allele Frequencies:
[0]=19.0
[1]=42.0
[2]=21.0
[3]=18.0
```

Figure 34. Initial Population with Unequal Trait and Allele Frequencies

Fitness Function

The fitness function for this genetic algorithm is largely configurable by the user. In order to model different situations, the user configurable parameters will have to be set in different ways. Each individual starts out with a fitness value of 100. The user can choose to have the fitness of one or more alleles fluctuate with the “AllelesToAlter" and "AlleleUpperBoundAlteration" parameters. The user can specify a comma separated list of alleles to alter in the “AllesToAlter”

parameter, and an upper bound on the percentage that the allele is allowed to fluctuate in the "AlleleUpperBoundAlteration". The user can also specify whether the allele will increase or decrease with the "IncreaseAllele" parameter. The user can choose to always increase the fitness value, always decrease the fitness value, or do both. If both increasing and decreasing are allowed, then it is randomly decided.

For each generation, random percentages between zero and the value specified in the "AlleleUpperBoundAlteration" will be generated for each allele that is allowed to fluctuate. In other words, the percentage that the allele fluctuates remains constant for all individuals in one generation. Similarly, whether the allele should increase or decrease or both remains constant in one generation. After all of the phases of the genetic algorithm for one generation are done, new fluctuation percentages are generated and whether an allele should increase or decrease is again randomly determined.

If an individual contains one of the alleles that is allowed to alter, then the fitness value for that individual will increase or decrease according to the specified percentage for that allele.

Reproduction

Reproduction is the first step in every generation. The classic roulette wheel style reproduction is performed. Each alignment in the population is given a percentage chance of survival equal to its relative fitness. Every alignment has at least some probability of being selected. No form of elitism is used.

Crossover

After reproduction, two individuals are randomly selected from the population for crossover. During crossover, the first allele from the first individual is combined with the second allele from the second individual. Conversely, the first allele from the second individual is combined with the second allele from the first individual. The two new individuals with new pairs of alleles are added to the population.

There are 3 situations where crossover will not be performed and the selected individuals will be added to the next population without any changes. First of all, crossover is only performed a percentage of the time. The user can specify the crossover percentage in the configuration parameters. The crossover percentage is typically around 70%. For every two individuals that are selected, a random percentage is generated to see if crossover should be performed or not. If the random percentage is greater than the specified crossover, then the two individuals are added to the population with no changes. Secondly, if the randomly chosen individuals are in fact the same individual, then the crossover will also have no effect. Thirdly, if the user has specified pairs of alleles that are not allowed to combine, then before the crossover of alleles, there is a check to make sure that the crossover of alleles will not result in invalid combinations of alleles. If the crossover would create an invalid combination, then the crossover is not performed and the individuals are added to the population with no changes.

Mutations

Mutation happens very rarely. If an individual is selected for mutation, than one of the alleles is randomly changed to another allele. Whether the first allele or the second allele will be changed is determined randomly. Then, a new random allele is generated and a check is done to make sure that the new combination of alleles is valid. A new random allele is generated until a valid combination is found.

Termination Conditions

After a population has gone through reproduction, crossover and mutation and a new generation has been formed, the new generation is evaluated to see if the algorithm should halt. The user can specify the maximum number of generations with the "GenerationSize" parameter. They can also specify the maximum number of unchanged rounds with the "UnchangedRoundsNeeded" parameter. Note that the "UnchangedRoundsNeeded" parameter should not be used when modeling the stable environment where the fitness value for all individuals is the same and does not change. The parameters can be used separately or in combination. The goal is allow the algorithm to reach a stable point.

Configuration Options

The user is allowed to specify a variety of parameters to influence the behavior of the genetic algorithm. The constructor for the main class takes a Property object as a parameter. Here is a sample file:

```
// 0 = Equal, 1 = Random
"InitialPopulation"="1";

// 0 = decrease, 1 = increase, 2 = both
"IncreaseAllele"="2";

"AlleleSize"="4";
"AllelesToAlter"="0";
"Nonpairs"="0-1, 0-2";
"AlleleUpperBoundAlteration"="20";
"PopulationSize"="100000";
"GenerationSize"="100";
"UnchangedRoundsNeeded"="5";
"ProbabilityCrossover"=".75";
"ProbabilityMutation"=".01";
```

Analysis of Results

Stable Environment

The first goal of this project was to model a stable environment where the selection on the alleles was constant. The genetic algorithm was set up to have four alleles with no environmental fluctuation. The population size was 100,000 and the genetic algorithm ran for 100 generations. The crossover probability was 75% while the mutation probability was 5%. The initial population for the first model was created so that the trait and allele frequencies were equal. Figure 35 shows the frequencies from the initial population:

```
---------------------------------------------------------------------
Generation 0:
---------------------------------------------------------------------
Trait Frequencies:
[00]=6.203999996185303
[01]=6.324999809265137
[02]=6.2689995765686035
[03]=6.296999931335449
[10]=6.177999973297119
[11]=6.153000354766846
[12]=6.289999961853027
[13]=6.233999729156494
[20]=6.079999923706055
[21]=6.1570000648498535
[22]=6.267999649047852
[23]=6.239999771118164
[30]=6.249000072479248
[31]=6.236000061035156
[32]=6.492000102996826
[33]=6.328000068664551

Allele Frequencies:
[0]=24.902999877929688
[1]=24.863000869750977
[2]=25.031999588012695
[3]=25.20199966430664

Statistics:
Random Allele Alterations: {3=0.0, 2=0.0, 0=0.0, 1=0.0}
Random Allele Increase Flags: {3=false, 2=false, 0=false, 1=false}
Fitness Sum: 1.0E7
Fitness Average: 100.0
Fitness Max: 100.0
---------------------------------------------------------------------
```

Figure 35. Initial Population of Stable Environment with Equal Frequencies

After running the genetic algorithm for 100 generations with equal selection and no fluctuation, the frequencies of the alleles changed slightly but overall they remained stable:

```
------------------------------------------------------------------------
Final Generation:
------------------------------------------------------------------------
Trait Frequencies:
[00]=6.109999656677246
[01]=6.050000190734863
[02]=5.986000061035156
[03]=5.876999855041504
[10]=6.395999908447266
[11]=6.2729997634887695
[12]=6.429000377655029
[13]=6.144999980926514
[20]=6.486000061035156
[21]=6.378000259399414
[22]=6.289000034332275
[23]=6.205999851226807
[30]=6.430999755859375
[31]=6.430000305175781
[32]=6.204999923706055
[33]=6.308999538421631

Allele Frequencies:
[0]=24.722999572753906
[1]=25.187000274658203
[2]=25.134000778198242
[3]=24.95599937438965

Statistics:
Random Allele Alterations: {3=0.0, 2=0.0, 0=0.0, 1=0.0}
Random Allele Increase Flags: {3=false, 2=false, 0=false, 1=false}
Fitness Sum: 1.0E7
Fitness Average: 100.0
Fitness Max: 100.0
------------------------------------------------------------------------
```

Figure 36. Final Population of Stable Environment with Equal Frequencies

In reality, it is very rare that alleles will all have the exact same frequency. Another test was done where the allele frequencies of the initial population were distributed randomly. It can be seen in Figure 37 that the initial frequencies do not alter the outcome of the test. The environment remains stable when selection is constant.

```
------------------------------------------------------------------
Generation 0:
------------------------------------------------------------------
Trait Frequencies:
[00]=5.480999946594238
[01]=5.546999931335449
[02]=5.572000026702881
[03]=5.427999973297119
[10]=9.914999961853027
[11]=9.888999938964844
[12]=10.104999542236328
[13]=9.98799991607666
[20]=5.51200008392334
[21]=5.629000186920166
[22]=5.684000015258789
[23]=5.585000038146973
[30]=3.9230000972747803
[31]=3.867999792098999
[32]=3.926999807357788
[33]=3.9469997882843018

Allele Frequencies:
[0]=23.42949867248535
[1]=32.415000915527344
[2]=23.849000930786133
[3]=20.306499481201172

Statistics:
Random Allele Alterations: {3=0.0, 2=0.0, 1=0.0, 0=0.0}
Random Allele Increase Flags: {}
Fitness Sum: 1.0E7
Fitness Average: 100.0
Fitness Max: 100.0
------------------------------------------------------------------
```

Figure 37. Initial Population of Stable Environment with Unequal Frequencies

```
-----------------------------------------------------------------
Final Generation:
-----------------------------------------------------------------
Trait Frequencies:
[00]=5.578999996185303
[01]=5.936999797821045
[02]=6.352999687194824
[03]=5.2779998779296875
[10]=9.076000213623047
[11]=9.86400032043457
[12]=10.473999977111816
[13]=8.54699993133545
[20]=5.442999839782715
[21]=5.778000354766846
[22]=6.091000080108643
[23]=4.995999813079834
[30]=4.065000057220459
[31]=4.253000259399414
[32]=4.5279998779296875
[33]=3.73799991607666

Allele Frequencies:
[0]=23.655000686645508
[1]=31.896499633789062
[2]=24.876998901367188
[3]=19.57149887084961

Statistics:
Random Allele Alterations: {3=0.0, 2=0.0, 1=0.0, 0=0.0}
Random Allele Increase Flags: {}
Fitness Sum: 1.0E7
Fitness Average: 100.0
Fitness Max: 100.0
-----------------------------------------------------------------
```

Figure 38. Final Population of Stable Environment with Unequal Frequencies

Fluctuating Selection Environment

The second goal was to model an environment where the selection on one or more alleles fluctuates over time while the other alleles remain constant. The first test illustrates the effects of having just one allele fluctuate. The genetic algorithm was set up to have four alleles where allele "3" was allowed to fluctuate over time while all other alleles remained constant. The population size was 100,000 and the genetic algorithm ran for 100 generations. The crossover

probability was 75% while the mutation probability was 0.125%. The initial population was created with a random allele distribution.

```
------------------------------------------------------------------
Generation 0:
------------------------------------------------------------------
Trait Frequencies:
[00]=16.809999465942383
[01]=8.697000503540039
[02]=9.23799991607666
[03]=11.503000259399414
[10]=8.730000495910645
[11]=0.7559999823570251
[12]=1.218999981880188
[13]=3.2150001525878906
[20]=9.255000114440918
[21]=1.2400000095367432
[22]=1.7769999504089355
[23]=3.734999895095825
[30]=11.110000610351562
[31]=3.2790000438690186
[32]=3.6630001068115234
[33]=5.773000240325928

Allele Frequencies:
[0]=46.076499938964844
[1]=13.946000099182129
[2]=15.95199966430664
[3]=24.02549934387207

Statistics:
Random Allele Alterations: {3=0.22, 2=0.0, 1=0.0, 0=0.0}
Random Allele Increase Flags: {3=false, 2=true, 1=true, 0=false}
Fitness Sum: 8970819.31999914
Fitness Average: 89.7081931999914
Fitness Max: 100.0
------------------------------------------------------------------
```

Figure 39. Initial Population with Randomly Distributed Frequencies

Although allele "3" is increasing and decreasing over time, the fact that it is fluctuating at all compared to the other alleles that are not fluctuating results in an overall decrease of allele "3". All of the other alleles increase to make up for the decrease in allele "3". Similarly, one can also see that the traits with allele "3" decrease while the other traits increase.

```
---------------------------------------------------------------------
Final Generation:
---------------------------------------------------------------------
Trait Frequencies:
[00]=31.865999221801758
[01]=9.92300033569336
[02]=13.26300048828125
[03]=2.267000198364258
[10]=9.173999786376953
[11]=2.758999824523926
[12]=3.929999828338623
[13]=0.6599999666213989
[20]=12.645999908447266
[21]=3.9099998474121094
[22]=5.269000053405762
[23]=0.9490000009536743
[30]=1.840000033378601
[31]=0.5950000286102295
[32]=0.7980000376701355
[33]=0.1509999930858612

Allele Frequencies:
[0]=56.42250061035156
[1]=16.854999542236328
[2]=23.017000198364258
[3]=3.7055001258850098

Statistics:
Random Allele Alterations: {3=0.18, 2=0.0, 1=0.0, 0=0.0}
Random Allele Increase Flags: {3=false, 2=false, 1=false, 0=true}
Fitness Sum: 9867091.240000026
Fitness Average: 98.67091240000026
Fitness Max: 100.0
---------------------------------------------------------------------
```

Figure 40. Final Population of an Environment with One Fluctuating Allele

The second test illustrates the effects of having multiple alleles fluctuate. The genetic algorithm was set up to have four alleles that were allowed to increase and decrease over time. The population size was 100,000 and the genetic algorithm ran for 100 generations. The crossover probability was 75% while the mutation probability was 1%. The initial population was created with a random allele distribution. One can see that the alleles fluctuate up and down in

cycles but that overall they remain within 5-10% of their original frequency over time.

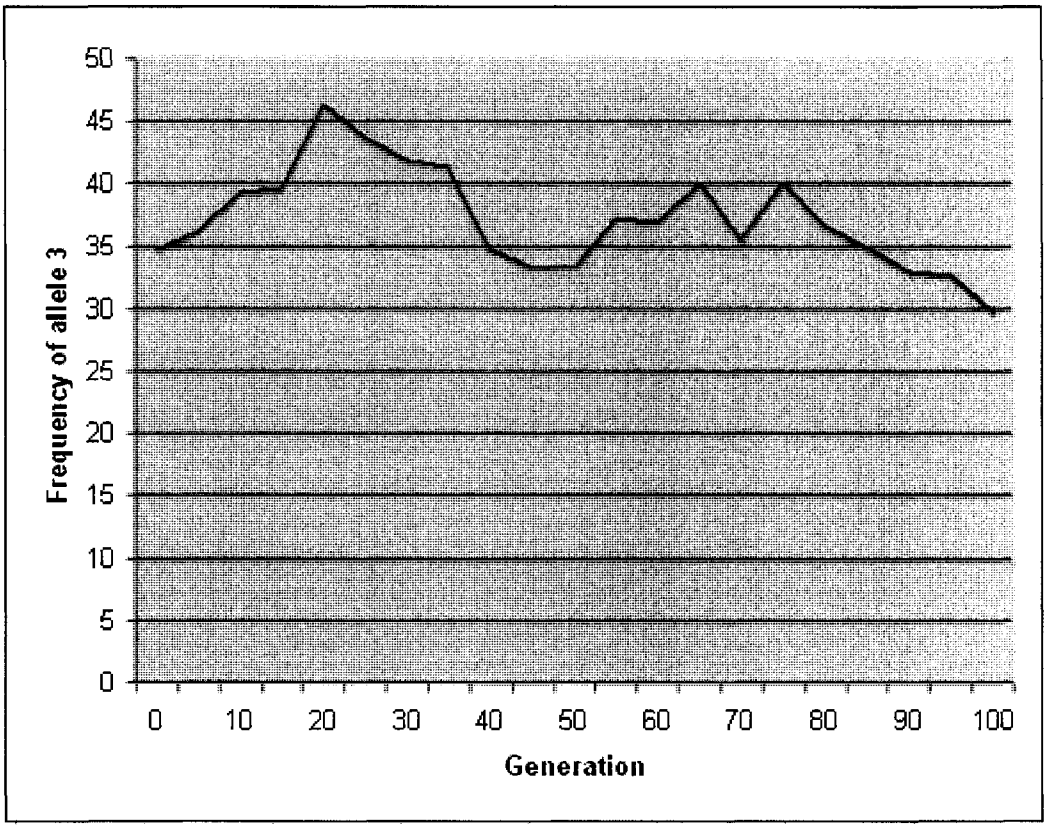


Figure 41. Fluctuation of Allele 3 When All Alleles Are Fluctuating

Hitch-Hiking Effect

The third goal was to model the hitch-hiking effect. If the selection on one or more alleles fluctuates over time and is not paired with certain other alleles, then the alleles that do not pair with the fluctuating allele will increase at a higher rate than the other non-fluctuating alleles. The genetic algorithm was set up to have four alleles where allele "3" was allowed to fluctuate over time while all other alleles remained constant. In addition, allele "3" was not allowed to pair with allele "0". All other combinations of alleles were valid. The population size

was 100,000 and the genetic algorithm ran for 100 generations. The crossover probability was 75% while the mutation probability was 0.125%. The initial population was created with a random allele distribution.

```
---------------------------------------------------------------------
Generation 0:
---------------------------------------------------------------------
Trait Frequencies:
[00]=16.270000457763672
[01]=10.418999671936035
[02]=8.32699966430664
[03]=0.0
[10]=9.486000061035156
[11]=9.440999984741211
[12]=7.331000328063965
[13]=2.8329999446868896
[20]=7.327000141143799
[21]=7.198999881744385
[22]=7.2669997215271
[23]=2.812999963760376
[30]=0.0
[31]=3.765000104904175
[32]=3.7129998207092285
[33]=3.809000253677368

Allele Frequencies:
[0]=34.04949951171875
[1]=29.957500457763672
[2]=25.62200164794922
[3]=10.371000289916992

Statistics:
Random Allele Alterations: {3=0.22, 2=0.0, 1=0.0, 0=0.0}
Random Allele Increase Flags: {3=false, 2=false, 1=true, 0=false}
Fitness Sum: 9562111.559999805
Fitness Average: 95.62111559999805
Fitness Max: 100.0
---------------------------------------------------------------------
```

Figure 42. Initial Population with Randomly Distributed Frequencies

```
------------------------------------------------------------------
Final Generation:
------------------------------------------------------------------
Trait Frequencies:
[00]=15.003999710083008
[01]=12.494999885559082
[02]=11.83899974822998
[03]=0.0
[10]=10.652999877929688
[11]=8.821000099182129
[12]=8.307000160217285
[13]=1.5210000276565552
[20]=9.723999977111816
[21]=8.388999938964844
[22]=7.652000427246094
[23]=1.3380000591278076
[30]=0.0
[31]=2.0260000228881836
[32]=1.8750001192092896
[33]=0.35600000619888306

Allele Frequencies:
[0]=37.359500885009766
[1]=30.516498565673828
[2]=28.38800048828125
[3]=3.7360000610351562

Statistics:
Random Allele Alterations: {3=0.04, 2=0.0, 1=0.0, 0=0.0}
Random Allele Increase Flags: {3=true, 2=false, 1=true, 0=false}
Fitness Sum: 1.0029944960000038E7
Fitness Average: 100.29944960000039
Fitness Max: 108.16
------------------------------------------------------------------
```

Figure 43. Final Population of an Environment with One Fluctuating Allele

In Figure 43, it can again be seen that the frequency of allele "3" has decreased and all of the pairs that have the fluctuating "3" have decreased. All of the other allele frequencies have increased, but allele "0" that was not paired with allele "3" has increased at the highest rate. Similarly, allele pairs that don't have allele "3" have increased, but allele pairs that have allele "0" have increased at the highest rate.

Conclusion

Genetic algorithms provide an effective way to model population genetics for different environments. They provide a stable yet flexible platform to model various scenarios. The genetic algorithm implementation presented in this thesis is able to model stable environments where the selection on the alleles remains constant and fluctuating environments where the selection on one or more alleles fluctuates. It is also able to model the hitch-hiking effect which shows how the alleles that do not pair with the fluctuating allele will increase at a higher rate than alleles that do pair with the fluctuating allele. The overall conclusion is that alleles that are affected by environmental fluctuation decrease in their frequency which also leads to a decrease in genetic diversity in the population.

One could expand the genetic algorithm implementation provided in this thesis to control the environment at a more detailed level. For example, instead of randomly selecting whether to increase or decrease the fitness of an individual per generation, one might want to support cycles of multiple generations that decrease and then increase. Also, one might want to set the upper bound of the fluctuation for each allele separately. Only fluctuating alleles with the lowest or highest frequencies would be another option.

References

Besemer, J., & Borodovsky, M. (1999). Heuristic approach to deriving models for gene finding. *Nucleic Acids Research, 27*(19), 3911-3920.

Besemer, J., & Borodovsky, M. (2005). GeneMark: Web software for gene finding in prokaryotes, eukaryotes and viruses. *Nucleic Acids Research, 33, Web Server Issue, W451-W454.*

Blanco E., Parra, G., & Guigó R. (2002). Using geneid to identify genes (A. Baxevanis, Ed). *Current Protocols in Bioinformatics, Unit 4.3.* New York: John Wiley & Sons Inc.

Blanco E., Parra G., Castellano S., Abril J.F., Burset M., Fustero X., et al. (2001). *Gene prediction in the post-genomic era.* Poster presented at the IX th meeting of the International Society for Matrix Biology, Copenhagen, Denmark.

Borodovsky, M., & McIninch, J. (1993). Genmark: Parallel gene recognition for both DNA strands. *Computers Chem., 17(*2), 123-133.

Burge, C.B., & Karlin, S. (1997). Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology, 268, 78-94.*

Burge, C.B., & Karlin, S. (1998). Finding the genes in genomic DNA. *Current Opinion in Structural Biology, 8, 346-354.*

Burge, C.B. (1998). GENSCAN command-line version README.txt documentation. (Version 1.0) [Computer Software]. Stanford University, Stanford, California.

Burge, C.B. (n.d.). *GENSCAN: Accuracy vs. exon probability.* Massachusetts Institute of Technology, Cambridge, Burge Group, Retrieved March 21, 2008 from http://genes.mit.edu/ExonProb.html

Buscema, M. (2004). Genetic doping algorithm (GenD): Theory and applications. *Expert Systems, 21*(2), 63-79.

Genome Bioinformatics Research Lab. (n.d.). *Geneid output format (Description).* Retrieved April 18, 2002, from http://genome.imim.es/software/geneid/docs/chapter4/formats_html/geneid.html

Genome Bioinformatics Research Lab. (n.d.). *Geneid home page main features.* Retrieved February 07, 2008, from http://genome.imim.es/software/geneid/

GeneMark: Background information. Georgia Institute of Technology, Atlanta, Borodovsky Group, Retrieved March 21, 2008, from http://exon.gatech.edu/GeneMark/background.html

Gillespie, J.H. (1972). Molecular evolution and polymorphism in a random environment. *Theoretical Population Biology, 4, 193-195.*

Gillespie, J.H. (1984). The interaction of genetic drift and mutation with selection in a fluctuating environment. *Theoretical Population Biology, 27, 222-237.*

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization & machine learning.* Reading, MA: Addison-Wesley Publishing Company, Inc.

Guigó R. (1998). Assembling genes from predicted exons in linear time with dynamic programming. *Journal of Computational Biology, 5, 681-702.*

Guigó R., Knudsen S., Drake N., & Smith, T.F. (1992). Prediction of gene structure. *Journal of Molecular Biology, 226, 141-157.*

Hernandez, D., Grass, R., & Appel, R. (2004). MoDEL: an efficient strategy for ungapped local multiple alignment. *Computational Biology and Chemistry, 28, 119–128.*

Horng, J.T., Wu, L.C., Lin C.M., & Yang, B.H. (2005). A genetic algorithm for multiple sequence alignment. *Soft Computing, 9, 407-420.*

Jacob, E., Sasikumar, R., & Nair, K.N.R. (2005). A fuzzy guided genetic algorithm for operon prediction. *Bioinformatics, 21, 1403-1407.*

Kimura, M. (1953). Process leading to quasi-fixation of genes in natural populations due to random fluctuation of selection intensities. *Genetics, 39, 280-295.*

Lukashin, A.V., & Borodovsky, M. (1998). GeneMark.hmm: New solutions for gene finding. *Nucleic Acids Research, 26*(4), 1107-1115.

Mathé, C., Sagot, M.F., Schiex, T., & Rouzze, P. (2002). Current methods of gene prediction, their strengths and weaknesses. *Nucleic Acids Research, 30, 4103-4117.*

Mendel, G. (1865). *Versuche ueber Pflanzen-Hybriden (Experiments in plant hybridization)*. Retrieved March 21, 2008, from http://www.mendelweb.org/Mendel.html

Moon, S.H., Choi, S.S. & Moon, B.R. (2006). A hybrid genetic search for multiple sequence alignment. *Proceedings of the 8th annual conference on Genetic and evolutionary computation GECCO '06 Publisher: ACM Press, July 2006.*

Mount, D.W. (2004). *Bioinformatics: Sequence and genome analysis* (2nd ed.). Cold Spring Harbor, NY: Cold Spring Harbor Laboratory Press.

Notredame, C., & Higgins, D.G. (1996). SAGA: Sequence alignment by genetic algorithm. *Nucleic Acids Research, 24, 8, 1515–1524.*

Parra G., Blanco E., & Guigó, R. (2000). Geneid in *Drosophila*. *Genome Research*, *10, 4, 511-515.*

Pavy, N., Rombauts, S., Déhais, P., Mathé, C., Ramana, D.V.V., Leroy, P., et al. (1999). Evaluation of gene prediction software using a genomic data set: Application to *Arabidopsis thaliana* sequences. *Bioinformatics, 15*(11), 887-899.

Radenbaugh, A.J. (2005). *Applications of genetic algorithms in computing architectures.* San José State University, Department of Computer Science, Course 247: Computer Architecture, Dr. Chun.

Radenbaugh, A.J. (2006). *Multiple sequence alignment using genetic algorithms: A review of the literature.* San José State University, Department of Computer Science, Course 100W: Technical Writing, Debra Caires, M.S.

Radenbaugh, A.J., Austin T. (2006). *Genetic algorithm for multiple sequence alignment: GAMSA.* San José State University, Department of Computer Science, Course 286: Advanced Bioinformatics, Dr. Khuri.

Seeluangsawat, P. & Chongstitvatana, P. (2005). A multiple objective evolutionary algorithm for multiple sequence alignment. *Proceedings of the 2005 conference on Genetic and evolutionary computation GECCO '05 Publisher: ACM Press, June 2005.*

Shyu, C., Sheneman, L., & Foster, J.A. (2004). Multiple sequence alignment with evolutionary computation. *Genetic Programming and Evolvable Machines, 5, 121–14.*

Steiner, U.K., Gaston, A.J., & Hipfner, J.M. (2002). *Hard and soft selection on chick growth rate in the thick-billed murre, and the implications of climate change.* Stanford University, Biological Sciences, Herrin Labs, Stanford, California.

Takahata, N., Ishii, K., & Matsuda, H. (1975). Effect of temporal fluctuation of selection coefficient on gene frequency in a population. *Genetics, 72*(11), 4541-4545.

Takahata, N. (1981). Genetic variability and rate of gene substitution in a finite population under mutation and fluctuating selection. *Genetics, 98, 427-440.*

The Arabidopsis Information Resource. (2008). *GBrowse visualization.* Retrieved March 21, 2008, from http://www.arabidopsis.org/cgi-bin/gbrowse/arabidopsis/

Uyar, H. Turgut, U., Sima, A., & Harmancı, E. (2006). Pairwise sequence comparison for fitness evaluation in evolutionary structural software testing. *Proceedings of the 8th annual conference on Genetic and evolutionary computation GECCO '06 Publisher: ACM Press, July 2006.*

Wang, C., & Lefkowitz, E.J. (2005). Genomic multiple sequence alignments: Refinement using a genetic algorithm. *BMC Bioinformatics, 6:200.*

Wikipedia (2008). *FASTA format.* Retrieved March 21, 2008, from http://en.wikipedia.org/wiki/FASTA_format

Wikipedia (2008). *Population genetics.* Retrieved March 21, 2008, from http://en.wikipedia.org/wiki/Population_genetics

Zhang, C., & Wong, A. (1997). A genetic algorithm for multiple sequence alignment. *Computer Application Bioscience, 13, 565-581.*

Appendix A: Amino Acid Test Cases

Amino Acid Hemoglobin Sequences – Population=100, Unchanged Rounds=20

```
[1]   -----GVLTDVQVALVKSSFEEFNANIPKNTHRFFTLVLEIAPGAKDLFSFLKGSSEVPQNNPDLQAHAGKVFKLTYEAAIQLQVNGAVASDATL
      KSLGSVHVSKG-VVDAHFPVVKEAILKTIKEVVGDKWSEELNTAWTIAYDELAIIIKKEMKDAA---

[2]   ---------VLSEGEWQLVLHVWAKVEADIAGHGQDILIRLFKHHPETLEKFDRFKHLKSEAEMKASEDLKKHGVTVLTALGAILKKKGHHEAEL
      KPLAQSHATKHKIPIKYLEFISEAIIHVLHSRHPADFGADAQGAMSKALELFRKDIAAKYKELGYQG

[3]   PIVDTGSVAPLSAAEKTKIRSAWAPVYSNYETSGVDILVKFFTSTPAAQEFFPKFKGLTTADQLKKSADVRWHAERIINAVNDAVVSMDDTEKMS
      MKLRDLSGKHAKSFQVDPQYFKVLAAVI------ADTVAAGDAGFEKLMSMICILLRSAY-------

[4]   ---------------VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNAL
      SALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR------

[5]   ---------------VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNAL
      SALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR------

[6]   ----------VQLSGEEKAAVLALWDKVNEEEVGGEALGRLLVVYPWTQRFFDSFGDLSNPGAVMGNPKVKAHGKKVLHSFGEGVHHLDNLKGTF
      AALSELHCDKLHVDPENFRLLGNVLVVVLARHFGKDFTPELQASYQKVVAGVANALAHKYH------

[7]   ----------VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTF
      ATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH------
```

Fitness: 1361.0 Time elapsed: 2:24
Compare with: 2775.0

Amino Acid Hemoglobin Sequences – Population=100, Unchanged Rounds=50

```
[1]   ----GVLTDVQVALVKSSFEEFNANIPKNTHRFFTLVLEIAPGAKDLFSFLKGSSEVPQNNPDLQAHAGKVFKLTYEAAIQLQVNGAVASDATLK
      SLGSVHVSKGVVDAHFPVVKEAILKTIKEVVGDKWSEELNTAWTIAYDELAIIIKKEMKDAA---

[2]   ---VLSEGEWQLVLHVWAKVEADIAGHGQDILIRLFKHHPETLEKFDRFKHLKSEAEMKASEDLKKHGVTVLTALGAILKKKGHHEAELKPLAQS
      HATKHKIPIKYLEFISEAIIHVLHSRH----PADFGADAQGAMSKALELFRKDIAAKYKELGYQG

[3]   PIVDTGSVAPLSAAEKTKIRSAWAPVYSNYETSGVDILVKFFTSTPAAQEFFPKFKGLTTADQLKKSADVRWHAERIINAVNDAVVSMDDTEKMS
      MKLRDLSGKHAKSFQVDPQYFKVLAAV----IADTVAAGDAGFEKLMSMICILLRSAY-------

[4]   ---------VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDL
      HAHKLRVDPVNFKLLSHCLLVTLAAHL----PAEFTPAVHASLDKFLASVSTVLTSKYR------

[5]   ---------VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDL
      HAHKLRVDPVNFKLLSHCLLVTLAAHL----PAEFTPAVHASLDKFLASVSTVLTSKYR------

[6]   ----VQLSGEEKAAVLALWDKVNEEEVGGEALGRLLVVYPWTQRFFDSFGDLSNPGAVMGNPKVKAHGKKVLHSFGEGVHHLDNLKGTFAALSEL
      HCDKLHVDPENFRLLGNVLVVVLARHF----GKDFTPELQASYQKVVAGVANALAHKYH------

[7]   ----VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSEL
      HCDKLHVDPENFRLLGNVLVCVLAHHF----GKEFTPPVQAAYQKVVAGVANALAHKYH------
```

Fitness: 1048.0 Time elapsed: 2:14
Compare with: 2775.0

Amino Acid Hemoglobin Sequences – Population=400, Unchanged Rounds=50

```
[1]   --------GVLTDVQVALVKSSFEEFNANIPKNTHRFFTLVLEIAPGAKDLFSFLK-GSSEVPQNNPDLQAHAGKVFKLTYEAAIQLQVNGAVAS
      DATLKSLGSVHVSKGVVDAHFPVVKEAILKTIKEVVGDKWSEELNTAWTIAYDELAIIIKKEMKDAA------

[2]   ---------VLSEGEWQLVLHVWAKVEADIAGHGQDILIRLFKHHPETLEKFDRFKHLKSEAEMKASEDLKKHGVTVLTALGAILKKKGHHEAEL
      KPLAQSHATKHKIPIKY----LEFISEAIIHVLHSRHPADFGADAQGAMSKALELFRKDIAAKYKELGYQG---

[3]   PIVDTGSVAPLSAAEKTKIRSAWAPVYSNYETSGVDILVKFFTSTPAAQEFFPKFKGLTTADQLKKSADVRWHAERIINAVNDAVVSMDDTEKMS
      MKLRDLSGKHAKSFQVD----PQYFKVLAAVIADTVAAGDAGFEKLMSMICILLR----------------SAY

[4]   ---------VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFP------HFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNAL
      SALSDLHAHKLRVDPVN----FKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR---------

[5]   ---------VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF------DLSHGSAQVKGHGKKVADALTNAVAHVDDMPNAL
      SALSDLHAHKLRVDPVN----FKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR---------

[6]   ----------VQLSGEEKAAVLALWDKVNEEEVGGEALGRLLVVYPWTQRFFDSFGDLSNPGAVMGNPKVKAHGKKVLHSFGEGVHHLDNLKGTF
      AALSELHCDKLHVDPEN----FRLLGNVLVVVLARHFGKDFTPELQASYQKVVAGVANALAHKYH---------

[7]   ----------VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTF
      ATLSELHCDKLHVDPEN----FRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH---------
```

Fitness: 2068.0 Time elapsed: 14:16
Compare with: 2775.0

Amino Acid Hemoglobin Sequences – ClustalW Result

```
[1]   --------GVLTDVQVALVKSSFEEFNANIPKNTHRFFTLVLEIAPGAKDLFSFLKGSSEVP--QNNPDLQAHAGKVFKLTYEAAIQLQVNGAVAS
      DATLKSLGSVHVSKGVVD-AHFPVVKEAILKTIKEVVGDKWSEELNTAWTIAYDELAIIIKKEMKDAA---

[2]   ---------VLSEGEWQLVLHVWAKVEADIAGHGQDILIRLFKHHPETLEKFDRFKHLKSEAEMKASEDLKKHGVTVLTALGAILKKKGH-----
      HEAELKPLAQSHATKHKIPIKYLEFISEAIIHVLHSRHPADFGADAQGAMSKALELFRKDIAAKYKELGYQG

[3]   PIVDTGSVAPLSAAEKTKIRSAWAPVYSNYETSGVDILVKFFTSTPAAQEFFPKFKGLTTADQLKKSADVRWHAERIINAVNDAVVSMDDT-EKM
      SMKLRDLSGKHAKSFQVDPQYFKVLAAVIADTVAAG---------DAGFEKLMSMICILLRSAY-------

[4]   ---------VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-----HGSAQVKGHGKKVADALTNAVAHVDD----M
      PNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR------

[5]   ---------VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-----HGSAQVKGHGKKVADALTNAVAHVDD----M
      PNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR------

[6]   --------VQLSGEEKAAVLALWDKV--NEEEVGGEALGRLLVVYPWTQRFFDSFGDLSNPGAVMGNPKVKAHGKKVLHSFGEGVHHLDN----L
      KGTFAALSELHCDKLHVDPENFRLLGNVLVVVLARHFGKDFTPELQASYQKVVAGVANALAHKYH------

[7]   --------VHLTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDN----L
      KGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH------
```

Fitness: 2775.0

Amino Acid Growth Hormone Sequences – Population=100, Unchanged Rounds=20

[1] -MAADSQTPWLLTFSLLCLLWPQEAGALPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS------I-QNA-QAAF-CFSETIPA
PTGKEEAQQRTDMELLRFSLLLIQSWLGPVQFLSRIFTNSLMFGTSDRVYEKLKDL-EEGIQALMQELEDGSPRIGQILKQTYDKFDANMRSDDA
LLKNYGLLSCFKKDLHKAETYLRVMKCRRFAESSCAF-

[2] -MAASPRNSVLLAFALLCLPWPQEVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYSIQNAQ------A--AFC-FSETIPA
PTGKDEAQQRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKLKDL-EEGIQALMRELEDGSPRAGQILKQTYDKFDTNLRSDDA
LLKNYGLLSCFKKDLHKAETYLRVMKCRRFVESSCAF-

[3] -MAAGPRNSVLLAFALLCLPWPQEVGTFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQ--------RYSIQNAQAAF-CFSETIPA
PTGKDEAQQRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKLKDL-EEGIQALMRELEDGSPRGGQILKQTYDKFDTNLRSDDA
LLKNYGLLSCFKKDLHKAETYLRVMKCRRFVESSCAF-

[4] -MAAGPRTSVLLAFGLLCLPWPQDVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS--------IQNAQAAFCFSETIPAP
-TGKDEAQQRSDMELLRFSLLLIQSWLGPVQLLSRVFTNSLVFGTSDRVYEKLRDL-EEGIQALMRELEDGSPRAGQILKQTYDKFDTNLRSDDA
LLKNYGLLSCFKKDLHKAETYLRVMKCRRFVESSCAF-

[5] -MAAGSWTAGLLAFALLCLPWPQEASAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYSIQNA-------QAAF—CFSETIPAP
TGKDEAQQRSDMELLRFSLLLIQSWLGPVQFLSRAFTNTLVFGTSDRVYEKLKDL-EEGIQALMRELEDGSPRVGQLLKQTYDKFDTNLRGDDAL
LKNYGLLSCFKKDLHKAETYLRVMKCRRFVESSCVF-

[6] -MAAGPRTSVLLAFTLLCLPWPQEAGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERTYIPEGQRYSIQN----AQAA---FC—FSETIPAP
TGKDEAQQRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKLKDL-EEGIQALMRELEDGSPRAGQILRQTYDKFDTNLRSDDAL
LKNYGLLSCFKKDLHKAETYLRVMKCRRFVESSCAF-

[7] -MAAGPRTSVLLAFALLCLPWTQEVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYSIQN------AQA---AFCFSETIPA
PTGKDEAQQRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKLKDL-EEGIQALMRELEDGSPRAGQILKQTYDKFDTNLRSDDA
LLKNYGLLSCFKKDLHKAETYLRVMKCRRFVESSCAF-

[8] -MAAGPRTSMLLAFALLCLPWTQEVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYSI------QNTQA--AF-CFSETIPA
PTGKDEAQQRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKLKDL-EEGIQALMRELEDGSPRAGQILKQTYDKFDTNMRSDDA
LLKNYGLLSCFKKDLHKAETYLRVMKCRRFVESSCAF-

[9] MMAAGPRTSLLLAFALLCLPWTQMVGAFPAMSLSGLFANAVLRAQHLHQLAADTFKEFERTYIPEGQR-YSIQ-----NTQV--AF-CFSETIPA
PTGKNEAQQKSDLELLRISLLLIQSWLGPLQFLSRVFSNSLVFGTSDRVYEKLKDL-EEGILALMRELEDGTPRAGQILKQTYDKFDTNMRSDDA

```
      LLKNYGLLSCFRKDLHKTETYLRVMKCRRFGEASCAF-

[10]  -MATGSRTSLLLAFGLLCLPWLQEGSAFPTIPLSRLFDNASLRAHRLHQLAFDTYQEFEEAYIPKEQ---K-----YSFLQNPQTSLCFSESIPT
      PSNREETQQKSNLELLRISLLLIQSWLEPVQFLRSVFANSLVYGASDSNVYDLLKDLEEGIQTLMGRLEDGSPRTGQIFKQTYSKFDTNSHNDDA
      LLKNYGLLYCFRKDMDKVETFLRIVQC-RSVEGSCGF-


Fitness: 38931.0  Time elapsed:  6:24
Compare with: 43520.0
```

Amino Acid Growth Hormone Sequences – Population=100, Unchanged Rounds=50

[1] -MAADSQTPWLLTFSLLCLLWPQEAGALPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNAQAAFCFSETIPAPTGKEEAQ
QR--TDMELLRFSLLLIQSWLGPVQFLSRIFTNSLMFGT-SDRVYEKLKDL-EE-GIQALMQELEDGSPRIGQILKQTYDKFDANMRSDDALLKN
YGL

[2] LSCFKKDLHKAETYLRVMKCRRFAESSCAF-MAASPRNSVLLAFALLCLPWPQEVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPE
GQRYS-IQNAQAAFCFSETIPAPTGKDEAQQRSDVELL--RFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYE-K-LKDLEEGIQALMRELED
GSPRAGQILKQTYDKFDTNLRSDDALLKNYGLLSCFKKDLHKAETYLRVMKCRRFVESSCAF

[3] -MAAGPRNSVLLAFALLCLPWPQEVGTFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNAQAAFCFSETIPAPTGKDEAQ
QR--SDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKL-KDL-EEGI-QALMRELEDGSPRGGQILKQTYDKFDTNLRSDDALLKN
YGLLSC

[4] FKKDLHKAETYLRVMKCRRFVESSCAF-MAAGPRTSVLLAFGLLCLPWPQDVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQR
YS-IQNAQAAFCFSETIPAPTGKDEAQQR--SDMELLRFSLLLIQSWLGPVQLLSRVFTNSLVFGTSDRVYE--KLRDLEE-GIQALMRELEDGS
PRAGQILKQTYDKFDTNLRSDDALLKNYGLLSCFKKDLHKAETYLRVMKCRRFVESSCAF

[5] -MAAGSWTAGLLAFALLCLPWPQEASAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNAQAAFCFSETIPAPTGKDEAQ
QR--SDMELLRFSLLLIQSWLGPVQFLSRAFTN-TLVFGTSDRVYEKLK-DLEE-GIQALMRELEDGSPRVGQLLKQTYDKFDTNLRGDDALLKN
YGLLSCFKKDLHKAETYLRVMKCRRFVESSCVF

[6] -MAAGPRTSVLLAFTLLCLPWPQEAGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERTYIPEGQRYS-IQNAQAAFCFSETIPAPTGKDEAQ
QR--SDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYE-K-LKDLEEGIQALMRELEDGSPRAGQILRQTYDKFDTNLRSDDALLKNY
GLLSCFKKDLHKAETYLRVMKCRRFVESSCAF

[7] -MAAGPRTSVLLAFALLCLPWTQEVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNAQAAFCFSETIPAPTGKDEAQ
QR--SDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYE-KLKDL-EEG-IQALMRELEDGSPRAGQILKQTYDKFDTNLRSDDALLKN
YGLLSCFKKDLHKAETYLRVMKCRRFVESSCAF

[8] -MAAGPRTSMLLAFALLCLPWTQEVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNTQAAFCFSETIPAPTGKDEAQ
QR--SDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKLK--D-LEEGIQALMRELEDGSPRAGQILKQTYDKFDTNMRSDDALLKN
YGLLSCFKKDLHKAETYLRVMKCRRFVESSCAF

[9] MMAAGPRTSLLLAFALLCLPWTQMVGAFPAMSLSGLFANAVLRAQHLHQLAADTFKEFERTYIPEGQRYS-IQNTQVAFCFSETIPAPTGKNEAQ
QK--SDLELLRISLLLIQSWLGPLQFLSRVFSNSLVFGTSDRVYEKLK---DLEEGILALMRELEDGTPRAGQILKQTYDKFDTNMRSDDALLKN

YGLLSCFRKDLHKTETYLRVMKCRRFGEASCAF

[10] -MATGSRTSLLLAFGLLCLPWLQEGSAFPTIPLSRLFDNASLRAHRLHQLAFDTYQEFEEAYIPKEQKYSFLQNPQTSLCFSESIPTP-SNREET
QQK-SNLELLRISLLLIQSWLEPVQFLRSVFANSLVYGASDSNVYDL-LKDLEEGIQTLMGRLEDGSPRTGQIFKQTYSKFDTNSHNDDALLKNY
GLLYCFRKDMDKVETFLRIVQ-CRSVEGSCGF

Fitness: 38728.0 Time elapsed: 18:39
Compare with: 43520.0

Amino Acid Growth Hormone Sequences – Population=400, Unchanged Rounds=50

[1] -MAADSQTPWLLTFSLLCLLWPQEAGALPAMPLSSLFAN-AVLRAQHLHQLAADTYKEFERAYIPEGQRYSIQNAQAAFCFSETIPAPTGKEEAQ
QRTDMELLRFSLLLIQSWLGPVQFLSRIFTNSLMFGTSDRVYEKLKDLEEGIQALMQELEDGS-PRIGQILKQTYDKFDANMRSDDALLKNYGLL
SCFKKDLHKAETYLRVMKCRRFAESSCAF

[2] -MAASPRNSVLLAFALLCLPWPQEVGAFPAMPLSSLFAN-AVLRAQHLHQLAADTYKEFERAYIPEGQRYSIQNAQAAFCFSETIPAPTGKDEAQ
QRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKLKDLEEGIQALMRELEDGS-PRAGQILKQTYDKFDTNLRSDDALLKNYGLL
SCFKKDLHKAETYLRVMKCRRFVESSCAF

[3] -MAAGPRNSVLLAFALLCLPWPQEVGTFPAMPLSSLFAN-AVLRAQHLHQLAADTYKEFERAYIPEGQRYSIQNAQAAFCFSETIPAPTGKDEAQ
QRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKLKDLEEGIQALMRELEDGS-PRGGQILKQTYDKFDTNLRSDDALLKNYGLL
SCFKKDLHKAETYLRVMKCRRFVESSCAF

[4] -MAAGPRTSVLLAFGLLCLPWPQDVGAFPAMPLSSLFAN-AVLRAQHLHQLAADTYKEFERAYIPEGQRYSIQNAQAAFCFSETIPAPTGKDEAQ
QRSDMELLRFSLLLIQSWLGPVQLLSRVFTNSLVFGTSDRVYEKLRDLEEGIQALMRELEDGS-PRAGQILKQTYDKFDTNLRSDDALLKNYGLL
SCFKKDLHKAETYLRVMKCRRFVESSCAF

[5] -MAAGSWTAGLLAFALLCLPWPQEASAFPAMPLSSLFAN-AVLRAQHLHQLAADTYKEFERAYIPEGQRYSIQNAQAAFCFSETIPAPTGKDEAQ
QRSDMELLRFSLLLIQSWLGPVQFLSRAFTNTLVFGTSDRVYEKLKDLEEGIQALMRELEDGS-PRVGQLLKQTYDKFDTNLRGDDALLKNYGLL
SCFKKDLHKAETYLRVMKCRRFVESSCVF

[6] -MAAGPRTSVLLAFTLLCLPWPQEAGAFPAMPLSSLFAN-AVLRAQHLHQLAADTYKEFERTYIPEGQRYSIQNAQAAFCFSETIPAPTGKDEAQ
QRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKLKDLEEGIQALMRELEDGS-PRAGQILRQTYDKFDTNLRSDDALLKNYGLL
SCFKKDLHKAETYLRVMKCRRFVESSCAF

[7] -MAAGPRTSVLLAFALLCLPWTQEVGAFPAMPLSSLFAN-AVLRAQHLHQLAADTYKEFERAYIPEGQRYSIQNAQAAFCFSETIPAPTGKDEAQ
QRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKLKDLEEGIQALMRELEDGS-PRAGQILKQTYDKFDTNLRSDDALLKNYGLL
SCFKKDLHKAETYLRVMKCRRFVESSCAF

[8] -MAAGPRTSMLLAFALLCLPWTQEVGAF-PAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYSIQNTQAAFCFSETIPAPTGKDEAQ
QRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDRVYEKLKDLEEGIQALMRELEDGS-PRAGQILKQTYDKFDTNMRSDDALLKNYGLL
SCFKKDLHKAETYLRVMKCRRFVESSCAF

[9] MMAAGPRTSLLLAFALLCLPWTQMVGAFPAMSLSGLFAN-AVLRAQHLHQLAADTFKEFERTYIPEGQRYSIQNTQVAFCFSETIPAPTGKNEAQ
QKSDLELLRISLLLIQSWLGPLQFLSRVFSNSLVFGTSDRVYEKLKDLEEGILALMRELEDGT-PRAGQILKQTYDKFDTNMRSDDALLKNYGLL

SCFRKDLHKTETYLRVMKCRRFGEASCAF-

[10] MATGSRTSLLLAFGLLCLPWLQEGSAFPTIPLSRLFDNASLRAHRLHQLAFDTYQEFEEAYIPKEQKYSFLQNPQTSLCFSESIPTPSNREETQQ
KSNLELLRISLLLIQSWLEPVQFLRSVFANSLVYGASDSNVYDLLKDLEEGIQTLMGRLEDGSPRTGQIFKQTYSKFDTNSHNDDALLKNYGLLY
CFRKDMDKVETFLRIVQCRS-VEGSCGF

Fitness: 40531.0 Time elapsed: 18:06
Compare with: 43520.0

Amino Acid Growth Hormone Sequences – ClustalW Result

```
[1]  -MAASPRNSVLLAFALLCLPWPQEVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNAQAAFCFSETIPAPTGKDEAQ
     QRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDR-VYEKLKDLEEGIQALMRELEDGSPRAGQILKQTYDKFDTNLRSDDALLKNYGLL
     SCFKKDLHKAETYLRVMKCRRFVESSCAF

[2]  -MAAGPRNSVLLAFALLCLPWPQEVGTFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNAQAAFCFSETIPAPTGKDEAQ
     QRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDR-VYEKLKDLEEGIQALMRELEDGSPRGGQILKQTYDKFDTNLRSDDALLKNYGLL
     SCFKKDLHKAETYLRVMKCRRFVESSCAF

[3]  -MAADSQTPWLLTFSLLCLLWPQEAGALPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNAQAAFCFSETIPAPTGKEEAQ
     QRTDMELLRFSLLLIQSWLGPVQFLSRIFTNSLMFGTSDR-VYEKLKDLEEGIQALMQELEDGSPRIGQILKQTYDKFDANMRSDDALLKNYGLL
     SCFKKDLHKAETYLRVMKCRRFAESSCAF

[4]  -MAAGSWTAGLLAFALLCLPWPQEASAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNAQAAFCFSETIPAPTGKDEAQ
     QRSDMELLRFSLLLIQSWLGPVQFLSRAFTNTLVFGTSDR-VYEKLKDLEEGIQALMRELEDGSPRVGQLLKQTYDKFDTNLRGDDALLKNYGLL
     SCFKKDLHKAETYLRVMKCRRFVESSCVF

[5]  -MAAGPRTSVLLAFTLLCLPWPQEAGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERTYIPEGQRYS-IQNAQAAFCFSETIPAPTGKDEAQ
     QRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDR-VYEKLKDLEEGIQALMRELEDGSPRAGQILRQTYDKFDTNLRSDDALLKNYGLL
     SCFKKDLHKAETYLRVMKCRRFVESSCAF

[6]  -MAAGPRTSVLLAFGLLCLPWPQDVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNAQAAFCFSETIPAPTGKDEAQ
     QRSDMELLRFSLLLIQSWLGPVQLLSRVFTNSLVFGTSDR-VYEKLRDLEEGIQALMRELEDGSPRAGQILKQTYDKFDTNLRSDDALLKNYGLL
     SCFKKDLHKAETYLRVMKCRRFVESSCAF

[7]  -MAAGPRTSVLLAFALLCLPWTQEVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNAQAAFCFSETIPAPTGKDEAQ
     QRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDR-VYEKLKDLEEGIQALMRELEDGSPRAGQILKQTYDKFDTNLRSDDALLKNYGLL
     SCFKKDLHKAETYLRVMKCRRFVESSCAF

[8]  -MAAGPRTSMLLAFALLCLPWTQEVGAFPAMPLSSLFANAVLRAQHLHQLAADTYKEFERAYIPEGQRYS-IQNTQAAFCFSETIPAPTGKDEAQ
     QRSDVELLRFSLLLIQSWLGPVQFLSRVFTNSLVFGTSDR-VYEKLKDLEEGIQALMRELEDGSPRAGQILKQTYDKFDTNMRSDDALLKNYGLL
     SCFKKDLHKAETYLRVMKCRRFVESSCAF
[9]  MMAAGPRTSLLLAFALLCLPWTQMVGAFPAMSLSGLFANAVLRAQHLHQLAADTFKEFERTYIPEGQRYS-IQNTQVAFCFSETIPAPTGKNEAQ
     QKSDLELLRISLLLIQSWLGPLQFLSRVFSNSLVFGTSDR-VYEKLKDLEEGILALMRELEDGTPRAGQILKQTYDKFDTNMRSDDALLKNYGLL
```

SCFRKDLHKTETYLRVMKCRRFGEASCAF

[10] -MATGSRTSLLLAFGLLCLPWLQEGSAFPTIPLSRLFDNASLRAHRLHQLAFDTYQEFEEAYIPKEQKYSFLQNPQTSLCFSESIPTPSNREETQ
QKSNLELLRISLLLIQSWLEPVQFLRSVFANSLVYGASDSNVYDLLKDLEEGIQTLMGRLEDGSPRTGQIFKQTYSKFDTNSHNDDALLKNYGLL
YCFRKDMDKVETFLRIVQCR-SVEGSCGF

Fitness: 43520.0

Appendix B: DNA Test Cases

DNA Simple Sequences

```
******
GAMSA Best Individual:
[1]   ATTGCCA-TT
[2]   ATGGCCA-TT
[3]   ATCCAATTTT
[4]   ATCTTC--TT
[5]   ATT-------
[6]   --GGCCA-T-
[7]   ATTG------
Fitness: -74.0

ClustalW Solution Individual:
[1]   ATCTTCTT--
[2]   ATCCAATTTT
[3]   ATT-------
[4]   --GGCCAT--
[5]   ATGGCCATT-
[6]   ATTGCCATT-
[7]   ------ATTG
Fitness: -84.0

Statistics:
Time Elapsed (HH:mm:ss:SS): 00:00:07:485
Population Size: 200
Number of Unchanged Rounds: 50
Number of Total Rounds: 68
Goodbye, and thank you for flying GAMSA.
******
```

DNA MYH16 Sequences

```
******
GAMSA Best Individual:
[1]   GAGCAGCTGAACAAGCTGATGACCACCCTCCACAGCACTGCACCCCATTTTGTCCGCTGTATTGTGCCCAATGAGTTTAAGCAGTCAG
[2]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTGTCCCCAATGAGTTTAAGCAATCGG
[3]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTGTCCCCAATGAGTTTAAGCAATCGG
[4]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
[5]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
[6]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
[7]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
[8]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAG--CCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
Fitness: 2322.0

ClustalW Solution Individual:
[1]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGC--CGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
[2]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
[3]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
[4]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTATCCCCAATGAGTTTAAGCAATCGG
[5]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTGTCCCCAATGAGTTTAAGCAATCGG
[6]   GAGCAGCTGAACAAGCTGATGACCACCCTCCATAGCACCGCACCCCATTTTGTCCGCTGTATTGTCCCCAATGAGTTTAAGCAATCGG
[7]   GAGCAGCTGAACAAGCTGATGACCACCCTCCACAGCACTGCACCCCATTTTGTCCGCTGTATTGTGCCCAATGAGTTTAAGCAGTCAG
Fitness: 1728.0

Statistics:
Time Elapsed (HH:mm:ss:SS): 00:00:16:360
Population Size: 200
Number of Unchanged Rounds: 50
Number of Total Rounds: 105
Goodbye, and thank you for flying GAMSA.
******
```

DNA Beta Globin Sequences

```
******
GAMSA Best Individual:

[1]   ----GTTTACGTTTGCTTCTGATTCTGTTGTGTTGACTTGCAACCTCAGAAACAGACATCATGGTGCACCT---GACTGATGCTGAGAAGGCTGC
      TGTCTC---TGGCCTGTGGGGAAAGGTGAACGCCGATGAAGTTGGTGGTGAGGCCCTGGGCAGGC

[2]   --------ACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAA--ACAGACACCATGGTGCATCTGACTCCTGA-GGA—GAAGTCTGCC
      GTT-A-C-TGCCCTGTGGGGCAAGGTGAACGTGGATGAAGTTGGTGGTGAGGCCCTGGGCAGGC

[3]   TAGGGCCCCTGCTGCTTACACCTGCTTTTGACACAACTTGCAGCTGGGTAAACACACATCATGGTGCATCT---GACTCTAGAAGAGAAGGCTAC
      T-TGTC-ACTGCCCTGTGGAGCAAGATGAGGGTGGCTGAAGTTGGTGTTGAAACCCTAGGCAGGC

[4]   --------ACACTTGCTTCTGACAC-ACCGTGCTCACTAGCAGCTGCACAAACACACACCATGC--T-GAC--T----GCTGAGGAGAAGGCTGC
      CGTCAC--C-GGCTTCTGGGGCAAGGTGAAAGTGGATGAAGTTGGTGCTGAGGCCCTGGGCAGGT

[5]   AGCTAGATTAGTTTCAGCATCATACTACTTTTGACACAGCTCTGTGTTCACAAGTAAACTTTCAAAATGGTGCACTTGACTTCTGAGGAGAAGAA
      CTGCATCACTACCATCTGGTCTAAGGTGCAGGTTGACCAGACTGGTGGTGAGGCCCTTGGCAGGT

Fitness: 53.0

ClustalW Solution Individual:

[1]   GTTTACGTTTGCTTC----TGATTC----TGTTGTGTTGACT---TGCAACCTCAGAAACAGACATCATGGTGCACCTGACTGATGCTGAGAAGG
      CTGCT-GTCTCTGGCCTGTGGGGAAAGGTGAACGCCGATGAAGTTGGTGGTGAGGCCCTGGGCAGGC

[2]   ----ACATTTGCTTC----TGACAC----AACTGTGTTCACT---AGCAACCTC—AAACAGACACCATGGTGCATCTGACTCCTGAGGAGAAGTC
      TGCC-GTTACTGCCCTGTGGGGCAAGGTGAACGTGGATGAAGTTGGTGGTGAGGCCCTGGGCAGGC

[3]   TAGGGCCCCTGCTGC----TTACACCTGCTTTTGACACAACT---TGCAGCTGGGTAAACACACATCATGGTGCATCTGACTCTAGAAGAGAAGG
      CTACTTGTCACTGCCCTGTGGAGCAAGATGAGGGTGGCTGAAGTTGGTGTTGAAACCCTAGGCAGGC

[4]   ----ACACTTGCTTC----TGACACACCGTGCTCAC-----T---AGCAGCTGCACAAACACACACCATG------CTGACTGCTGAGGAGAAGG
      CTGCC-GTCACCGGCTTCTGGGGCAAGGTGAAAGTGGATGAAGTTGGTGCTGAGGCCCTGGGCAGGT
```

```
[5]    AGCTAGATTAGTTTCAGCATCATAC-TACTTTTGACACAGCTCTGTGTTCACAAGTAAACTTTCAAAATGGTGCACTTGACTTCTGAGGAGAAGA
       ACTGC-ATCACTACCATCTGGTCTAAGGTGCAGGTTGACCAGACTGGTGGTGAGGCCCTTGGCAGGT

Fitness: 352.0

Statistics:
Time Elapsed (HH:mm:ss:SS): 00:00:26:407
Population Size: 200
Number of Unchanged Rounds: 70
Number of Total Rounds: 200
Goodbye, and thank you for flying GAMSA.
******
```

DNA HIV Sequences

******
GAMSA Best Individual:

[1] G-AAGGAAATCAGATACATGTTATAAAGCCAGTTT-TACCAAAGCTAAACAATCTGTTTGAAT-ATGC-GGTGTCAGAGGAAAA-TGGATGTTGG
---CTCTTT-CA--CCCATCCT---G--GCA-TATGTGTAT-TTATAAAAGAATGTTTGAA-AGTGAAAACAAAATCCTGTCCAAAGAAGGTGTT
ATCCA--TTTTTTGGAGCTG---T-ATGAAACAAAGATTCTTCCATTTTCACCAGAAT-TTTCTGAGTTTATTATTGGACCAT-TAATGGATGCG
CTTTCAGAGAGCTCTCTGTATAGCAGGT-CCCCAGGCCAGCCAATAGGAAGCTG-TTCTCCATTGGGACTGAAATTAC-AGAAGTTTTTAGTCAC
TTATATTTCTCTTCTTCCAGAAG-A--A-ATAAAGA-GTAGCTTCCTATTGAAGTTTATTCGGAAGATGACAAG--TAGGCA-TTGG-T-GTGCT
-G-TTC-CCATTTT-GTTTCTATCTAAG-GCTTT---G-GCAAATGTCCCAA-GACA--TA-AGGCCC--TGG-GTA-TAGATGGGCTTC-TT-G
CTC-TC----AGGGA-TGTTATTCATTG--C-ACTATGATCA-C-ACATCAGATTCTCCTGAGAGGGGCAGCCCAATGCTACCTTCTTCAAACAG
CTATGAATTTGCTAGATGTGGAGAAAGTGTCACTTTCT

[2] G-AAGGAAATCAGATACATGTTATAAAGCCAGTTT-TACCAAAGCTAAACAATCTGTTTGAATATGCGGTGTCAGAGGAAAATGGATGTTGGCTC
TTTCACCCATCCTGGCATATGT------GTATTTATAAAAGAATGTTTGAAAGTGAAAAC-A-AAAT-C-CTGT-CCAAAGAAGGTGT-TATCCA
TTT-T-TTGG-AGC--TGTATG-AAA-CAAAGATTCTT-CCATTTTCACCAGAATTTT-CTGAG--TTTA-TTATTGGACCA-TTAATGGATGCG
CTTTCAGAGAGCTCTCTGTATAGCAGGTCCCCAGGCC-AGCCAATAGGAAGCTGTTCT-CCATTGGGACTGAAATTACAGA-AGTTTTTAGTCAC
TTATATTTCTCTTCTTCCAGAAG-AAATAAAGAGTAGCTTC-CTATTGAAGTTTATTCGGAAG-A--TGACAA-GT-AGGCA-TTGG--TGTG-C
TG-T-TCCC-AT-TT--TGTTTC---TATCTA-A----------GGCTT-T-GGCAAAT---GTCCCAA-GACATAAGGCCCTGGGTATAGATG
GGCTTCTTGCTCTCA-GGGATGTTATTCATTG-CACTATGATCACACATCAGATTCTCCTGAGAGGGGCAGCCCAATGCTACCTTCTTCAAACAG
CTATGAATTTGCTAGATGTGGAGAAAGTGTCACTTTCT

[3] -GACAGTCTAGTGGTTCGAGTA--AT-TTTGGA-G--G-TTT-CCCCACA--GCAAG--TCACTCTCCTTTTC--AGCCCCAAACTACAGGTGG-
AAGTGCTG--CATCAGTAAA-TG-C---TAATTTTGCTCATTTTGATAACTTC-CCCAAATCCTCCAGTGCTGATTTTGGAACCT-TCA-ATACT
TC-CCAG-AGTCATCAAACAGC-ATC-AGCTGTTAGT-AAAGTTTCAACGAACAAAGCTGGTTTACAGACTGCAGACAAATATGCAGCACTTGCT
AATTTAGAC-AATAT-CTTCAG-TGC-CGGGCAAGGTGGTG-ATCAGGGA----AGTGGCTTTGGGACCACAGGT-AAAGCTCCTGTTGGTTC-T
GTGGTTTCAG-TTCCCAGTCA-G-TCAAGTGCATCT-TCAG-ACAAGTA---TGCA-GCTC-TGGCAG-AACTAGACA--G-CGT-T-TTCAGTT
CTGCAGC-C-ACCTCC-AGTAATGCGTATACTTC----CAC-A-AGTAATGCTAGCAGCAA---TGTTT-TTG-G-AACAG—TGCCAGTGGTTGC
--TTC----TGCAC--AGAC-ACAGCCTGCTTCA-TCAAGTGTGCCTGC-TCCA--TT-T-GGAGCTACGCCTTC-CACAAATCCATTTGTTGCT
GCTGCTGGTCCTTCTGTGGCATCTTCTACAAACCCAT

[4] ---GCGACACCAGCTGCACTGG-TCAGGGCCC-CAGCAAGAAGGCAGCCAAGCACAAGGCAGCTGAGGTGGC---CCTCAAACACC-TCAAAGGG
GGGAGCATGCTGGAGCCG-GCCC-TG--GAGGACAGCAGTT-CTTTTTCTCCCCTA--GA-CTC-TTC-ACTGCCTGAGGACATTCCGGTTTTTA
CTGCTGCAGCAGCTGCTACCCC--AG--TTCCA-TCT-GTAG-TCCTAAC-CAGGAGCCCC-CCCATGGAA-CT---GCAGCC-CCCTGTCTCCC
-C-TCAGCAGTCTGAGTGCAACCCCGTTGGTG-CTCTGCAGGAGCTGGTGG-TGCAGAAAGGCTGGCGGTTGCC-GGAG—TACACAGTGACCCAG

GAGTCTGGGC-CAGCCCAC-CG-CA-AAGAATTC-ACCATGACCTGTCG-AGTGG-AGCGTT-TC-ATTGAGAT-TGGGAGTGGCACTTCCAAAA
A-ATTGGCA-AAGC-G--GAATGCGGCGGCC-AAAATGCTGCT-TCGAGTGCAC-ACGG----TGCCTCTG-GATGCCCGGGAT--GGCA-ATGA
GGTGGAG-CCTGAT-GATGACCACTTCTCCATTGGTGTGGGCTCCCGCCTGGATGGTCTTCGAAACCGGGGCCCAGGTTGCACCTGGGATTCTCT
ACGAAATTCAGTAGGAGAGAAGATCCTGTCCCTCCGC

[5] ------AGTGCC-CAGCGAGACAGCCCCGGCCTCG--GTTTTTCCCCAGAATGGA-GCCGCCC-GTA-GGGCAGT-G-GCAGCCCAGCCAGGACG
--G-AAGCGAAAA-TCGAATTGTT-TGGGCACTGATGAGGACTCCCAGGACAGCTCTGATGGAATACCGTCAGCACCACGCATGACTGGCAGCC-
TGG-T-GTCTGATCGAAGCCACGACGACATCGTCACCCGGATGAAGAACATTGAGTGCAT-TG-AGCTGGGCCGGCACCGCCTCA-AGCCGTGGT
ACTTCTCCC--CGTACCCACAGGAACTCAC-CACATTGCCTGTCCTCTA--CCTGTGCGAGT-TCTGCCTCAA-GT-AC-GG-C-CGTAGTCTCA
AGTGTCT--TCAGCGTCAT-TTGACC-AA--GTGT-GACCT-ACGACATCC-TCCAGGCAATGA-G-ATTTACCGCAAGGGCACCATCTCCTT-C
TT--TGAGATTG-ATGGACGTAA-GAACAAGAG--T-T--ATTCCCAGAACCTGTGTCTTTT-GGCCAA-GTG-TTTCC-TT-GACC-ATA-AGA
CAC-TGTACTATGACACA-GACCC--T-TTCCTCTTCTAC-GT-CATGACAGAGTATGACTGTAAGGGCTTCCACATCGTGGGCTACT-TC-TCC
AAGGAGAAAGAATCAACGGAAGACTACAATGTGGCC—

[6] AAGTGTCATGTACACAA--GCATTTCTCAGA--TACTTGGGCAGAATAGCCCTGCCATTGTCATATGCAA-AGTCGATGAGAATATGACCCAAAG
GACACTGGTCACCAA-CGC-AGC--CATGCAAGGGATAG-GATT-CAACA-TTG-CCCAGGTGCTGGGGCA-GCA-TGCG—GGCTTGGAGAAGTA
CCCCATTTGGA-A-A-GCAC--CTCAGACT-TTGCCCCTCGGCTTAGAATCCTCCATCCC-CTTGTGTTTACCTTC--CACCTCTGAC-AGCGTG
GCCACCCTGGGAGGTAGCAAGCGAATGCTT-TCTCCAGCCAGTAGCTTG-GAG-CTCTTCATGGAAACCAAGCA-GCAGAAAAGGGTCAA-AGAA
GAAAAGATGTACGGACAG-ATTGTGG-AGGA-G-CT-T-AGTGCTGTGGAGCTGACCA-ACTCAGAC-ATCAAAAAGGACCTCTCCC-GCCCCCA
GA-AACCCC---AGCTGG-T-TCGACAAGG-A-TGTGCTTCTGAGCCAAAAGA--TGGCTTGCAGTCAGGGTCAT-CTT—CCTTCTCCTCGCTGT
CGCC-CTCC-TCATCTCAAGACTATCCTTCTGTTAGCCCGTCTTCCAGGGAGCCATTCCCGCCCAGCA-AGGAGATGCTTTCCGGTTCCCGGGCA
CCACTTCCGGGGCAGAAGTCCAGTGGGCCTTC

Fitness: -6557.0

ClustalW Solution Individual:

[1] GAAGGAAATCAGATACATGTTATAAAGCCAGTTTTACCAAAGCTAAACAATCTGTTTGAATATGCGGTGTCAGAGGAAAATGGATGTTGGCTCTT
TCACCCATCCTGGCATATGTGTATTTATAAAAGAATGTTTGAAAGTGAAA-ACAAAATCCTGTCCAAAGAAGGTG-TTATCCATTT-TTTGGAGC
TGTATGAAACAAAGATTCTTCCATTTTCAC-CAGAATTTTCTGAGTTTATTATTGGACCATTAATGGATGCGCT--TTCAGAG—AGCTCTCTGTA
TAGCA-GGT--CCCCAGGCCAGCCAATAG--GAAGCTGTTCTCCATTGGGACTGAAATT-ACA--GAAGTTTTTAGTCACTTATATTTCTC—TTC
TT-CCAG-AAGAAATAAAGAGTAGCTTCC---TATTGAAGTTTATTCGGAAGATGACAAGTAGGC-ATTGGTGT-GCTGTT-CCCATTTTGTTTC
T-ATCTAAGGCTTTGGCAAATGTCCCAAGACATAAGGCCCTGGGTATAGATGGGCTTCTTGCTCTCAGGGATGTTATTCATTGCACTATGATCAC
ACATCAGATTCTCCTGAGAGGGGCAGCCCAATGCTACCTTCTTCAAACAGCTATGAATTTGCTAGATGTGGAGAAAGTGTCACTTTCT

[2] GAAGGAAATCAGATACATGTTATAAAGCCAGTTTTACCAAAGCTAAACAATCTGTTTGAATATGCGGTGTCAGAGGAAAATGGATGTTGGCTCTT

TCACCCATCCTGGCATATGTGTATTTATAAAAGAATGTTTGAAAGTGAAA-ACAAAATCCTGTCCAAAGAAGGTG-TTATCCATTT-TTTGGAGC
TGTATGAAACAAAGATTCTTCCATTTTCAC-CAGAATTTTCTGAGTTTATTATTGGACCATTAATGGATGCGCT--TTCAGAG—AGCTCTCTGTA
TAGCA-GGT--CCCCAGGCCAGCCAATAG--GAAGCTGTTCTCCATTGGGACTGAAATT-ACA--GAAGTTTTTAGTCACTTATATTTCTC—TTC
TT-CCAG-AAGAAATAAAGAGTAGCTTCC---TATTGAAGTTTATTCGGAAGATGACAAGTAGGC-ATTGGTGT-GCTGTT-CCCATTTTGTTTC
T-ATCTAAGGCTTTGGCAAATGTCCCAAGACATAAGGCCCTGGGTATAGATGGGCTTCTTGCTCTCAGGGATGTTATTCATTGCACTATGATCAC
ACATCAGATTCTCCTGAGAGGGGCAGCCCAATGCTACCTTCTTCAAACAGCTATGAATTTGCTAGATGTGGAGAAAGTGTCACTTTCT

[3] GACAGTCTAGTGGTTCGAGTAATTTTGGAGGTTTCCCCACAGCAAGTCACTCTC-----CTTTTCAGCCCCAAACTACAGGTGGAAGTGCTGCAT
-CAGTAA---ATGCTAATT-----TTGCTCA----TTTTGATAACTTCCCCAAATCCTCCAGTGCTGATTTTGGAACCTTCAATACTTCCCAGAG
TCATCAAACAGCATCAGCTGTTAG-TAAAGTTTCAACGAACAAAGCTGGTTTACAGACTGCAGACAAATATGCAGCACTTGCT--AATT---TAG
ACAATA-TCT--TCAGTGCCGGGCAAGGTG--GTGATCAGGGA--AGTGGCTTTGGGACC-ACAGGTAAAGCTCCTGTTGGTTCTGTGGTTT—CA
GTT-CCCA-GT-CAGTCAAGTGCATCTTCA---GACAAGTATGCAGCTCTGGCAGAACTAGACAGC-GTTTTCAGTTCTGCAGCCACCTCCAGTA
AT-GCGTATACTTCCAC-AAGTAATGCTAGCAGCAATGTTTTTGGAACAGTGCCAGTGGTTGCTTCTGCACA-GACACAGCCTGCTTCATCAA-G
TGTGCCTGCTCCATTTGG-AGCTACGCCTTCCACAAATCCATTTGTTGCTGCTGCTGGTCCTTCTGTGGCATCTTCTACA—AACCCAT

[4] GCGACACCAGCTGCACTGGTCAGGGCCCCAGCAAGAAGGCAGCCAAGCACAAGGCAGCTGAGGTGGCCCTCAAACACCTCAAAG-GGGGGAGCAT
GCTGGAGCCGGCCCTGGAG-----GACAGCAGTTCTTTTTCTCCCCTAGACTCTTCACTGCCTGAGGACATTCCGGTTTTTACTGCTGCAGCAGC
TGC-TACCCCAGTTCCATCTGTAGTCCTAACCAGGAGCCCCCCCATGGAACTGCAGCCCCCTGTCTCCCCTCAGCAGTCTGAG—TGCAACCCCGT
TGGTGCTCT--GCAGGAGCTGGTGGTGCAGAAAGGCTGGCGGTTGCCGGAGTACACAGTGACCCAGGAGTCTGGGCCAGCCCACCGCAAAG—AAT
TC-ACCATGACCTGTCGAGTGGAGCGTTT---CATTGAGA-TTGGGAGTGGCACTTCCAAAAAATTGGCAAAGCGGAATGCGGCGGCCAAAATGC
T-GCTTCGAG-TGCACACGGTGCCTCTGGATGCCCGGGATGGCAATGAGGTGGA--GCCTGATGATGACCA-CTTCTCCATTGGTGTGGGCTCCC
G-CCTGGATGGTCTTCGAAACCGGGGCCCA-GGTTGCACCTGGGATTCTCTACGAAATTCAGTAGGAGAGAAGATCCTGTCCCTCCGC

[5] AGTGCCCAGCGAGACAGCCCCGGCCTCGGTTTTTCCCCAGAATGGAGCCGCCCG-----TAGGGCAGTGGCAGCCCAGCCAGGACGGAAGCGAAA
—ATCGAATTGTTTGGGCA-----CTGATGAGGACTCCCAGGACAGCTCTGATGGAATACCGTCAGCACCACGCA-TGACTGGCAGCCTGGTGTCT
GATCGAAGCCACGACGACATCGTCACCCGGATGAAGAACATTGAGTGCATTGAGCTGGGCCGGCACCGCCTCAAGCCGTGGT—ACTTCTCCCCGT
A----CCC—ACAGGAACTCACCACATTGCCTGTCCTCTACCTGTGCGAGTTCTGCCTCAAGTACGGCCGTAGTCTCAAGTGTCTTCAGCGTCATT
TGACCAAGTGTGACCTACGACATCCTCCAGGCAATGAGATTTACCGCAAGGGCACCATCTCCTTCTTTGAGATTGATGGACGTAAGAACAAGAGT
TATTCCCAG---AACCTGTGTCTTTTGGCCAAGTGTTTCCTTGACCATAAGACACTG-TACTATGACACAGACCCTTTCCTCTTCTACGTCATGA
CAGAGTATGACTGTAAGGGCTTCCACATCGTGGGCTACTTCTCCAAGGAGAAAGAATCAACGGAAGACTACAATGTGG-------CC

[6] GAAGTGTCATGTACACAAGCATTTCTCAGATACTTGGGCAGAATAGCCCTGCCATTGTCATATGCAAAGTCGATGAGAATATGACCCAAAGGACA
CTGGTCACCAACGCAGCCATGCAAGGGATAGGATTCAACATTGCCCAGGTGCTGGGGCAGCATGCGGGCTTGGAGAAGTACCCCATTTGGAAAGC
ACCTCAGACTTTGCCCCTCGGCTTAGAATCCTCCATCCCCTTGTGTTTACCTTCCACCTCTGACAGCGTGGCCACCCTGGGAGGTAGCAAGCGAA
TGCTTTCTCCAGCCAGTAGCTTGGAGCTCTTCATGGAAACCAAGCAGCAGAAAAGGGTCAAAGAAGAAAAGATGTACGGACAGATTGTGGAGG-A
GCTTAGTGCTGTGGAGCTGACCAA----CTCAGACATCAAAAAGGACCTCTCCCGCCCCCAGAAACC---CCAGCTGGTTCGACAAGGATGTGCT

```
TCTGAGCCAAAAGATGGCTTGCAGTCAGGGTCATCTTCCTTCTCCTCGCTGTCGCCCTCCTCATCTCAAGACTATCCTTCTGTTAGCCCGTCTTC
CAGGGAGCCATTCCCGCCCAG---------CAAGGAGATGCTTTCCGGTTCCCGGGCACCACTTCCGGGGCAGAAGTCCAGTGGGCCTTC
```

```
Fitness: -3571.0

Statistics:
Time Elapsed (HH:mm:ss:SS): 00:01:01:986
Population Size: 200
Number of Unchanged Rounds: 70
Number of Total Rounds: 200
Goodbye, and thank you for flying GAMSA.
******
```

DNA BRCA1 Sequences - Maximum Number of Rounds = 200

******
GAMSA Best Individual:

[1] -----CAGAGTATTTCTATTTCAAATGTGTGTGTGGAGCCATGTGGCACAGATGCTCATGCCAG-CTCATTACAGCCTGAGACCAGCAGTTTATT
GCTCATTGAAGACAGAATGAATGCAGAAAAGG-CTGAATTCT-GTAA-TAAAAGCAAACAG-CC--TGGCATA-GCAGTGAGCCAGCAGAGCAGA
TGGGCTGCAAGT-AAAGG-AACATGTAAC-GACAGGCA----------GGTTCCCAGCACTGGGGAAAAGGTAGGTCCAAACGCT-GACTCC
CTTAGTGATAGAGA-G-AA--GT-GGAC-TCAC--C-CGCAAAGTCTGTGCCCTGAGAATTCTGGAGCTACCACCGATGTTCCTTGGATAACACT
AAA--TAGCAGCGTTCAGAAAGTTAATGAGT------GGTTTTCCA----------GAA-CTG---GTGAAATGTTAACTTCTGACAGCGCAT
CTGCCAGGAGGCACGAGTC--AA--ATGCTGAAGCAGCTGTTGTGTTGGAAGTTTCAA--ACGAAGTGGATGG-GGGTTTTA-GTTCTTC---AA
GGAAAACAGACTTAGTAACCCC-CGAC---CCCCA--TCATACTTTAA-TGTGTAAAAGT-GGAAGAGACTTCTCCA-AACCAGTAGAGGATAAT
ATCAGTGATAAAATATTTGGGAAATCCTAT-CAGAGAAAGGGAAGCCGCCCTCACCTGAACCAT-GTGACTGAAATTATAGGCACATTTATTACA
GAACCACAGATAACACAAGAG

[2] --CCGCGTATTTCTG-TT-GCAAACGTGCACGT-GGAGCCGTGTGGCACAGATGCTCGTGCCA-GCTCATTACAGCGTGGGACCCGCAGTTTATT
GTTCACTGAGGACAGACTGGATGCAGAAAAGGCTGAATTCTGTGATAGAAGCAAACAGTCTGGCG-CAG-CAGTGAGC-CAG--C-AGAGCAGAT
GGGCTGA-C-AGT-AAAGAAACATGTAA-TGGCAGGCCGGTTCC---CCGCA-CTGA--GGGAAAGGCAGATCCAAATGTGGATTCC-CTCTGT-
GGTAG-AAAGCA-GTGGAATCATCCGAAAAGC-CTGT-GC-CCTGAGAATTCTGGAGCTACCACTGACGTTCCTTGGATAACACTGAATAGCAGC
ATTC-AGAAAGTGAATGAGTGGTTTTCCAGA--AC---TGG-T----------G-AAAT-GTTAA-CTTCTGAC-AATGCATCTGACAGG
AGGCCTGCGTCAAATGCAGAAGCTG-C-TGTTGTGTTAGAAGTTTCAAATGAAGTGGATGGATGTTTCAGTTCTTCAAAGAAAAT-AGACTTAG-
TTGCCC-CTGA-TCCCGA-TAATGCTGTAATGTGT-ACAAGTGGA-AGA--GACT-TCTC-CAAGCCAGT-AGAGA-ATATTATCAACGATAAAA
TATTTGGGAAAACCTATCAGAGAAAGGG--AAGCCGC-CCTC-----AC-TTGAACC-A-TGTGA-CTGAAATTATAGGCACATTTACTACAGAA
CCACAGATTATACAAGAG

[3] ---CAGGGTATTTCTGTTTCAAACTTGCAT-GTGGAGCCATGTGGCACAAATACTCATGCCA-GCTCATTACAGCATGAGAACAGCAGTTTA-TT
ACTCA----CT-AA--ACACAGAATGAATGT-AGAA-AAG-GCTGAAATCT-GTAATAAC-AGCAAACAGCCTGG-CT-TAGCAAGGAG-CCAAC
AGAGCA-GATGGGCTGAAAGTAAGGAAACATGT-AATGATAGGCAGATTCCCAGCAC----AGAGA-AA--AAGGTAGTTGTGAA-TGCTGATCT
CCTGTGTGGGAGAAAAGAACTGAATAAACAGAAACCTCCACACTCTGATAGTCCTAGAGATTCCCAAGATGTTCCTTGGATAACACTGAATA-GT
AGC-A-TACGGAAAGTTAATGAGTGGT--TTTC-----CAGAAGTG-ACGAA--ATATTAACTTC-T-GATGA-TT-CACATGACAGAGGATCTG
-AATTGAATACT-GAAG-TAGGTG--GTGCAGTAGAAGTTCCAAATGAAGTGGGTGAATATTCTG-GT-TCT--TCTGAGAAA-ATAGACT-TAA
TGGCCAGTGAT-CC-TCAGGATGCT-TTCA-T-----ATGTGAAAG-T-GAAAGAG-TCC-ACACC-AAGCCAGTAGGAGGTAATATCGAAGATA
AAATATTTGGA-AAA-ACCTAT-C-GGAGG-AAGGCAA-GCCTCCCTAAGGTGAGCCACACAACTGAAGTTCTAACTATAGGAGCGTGTGCTATA
GAACCTCAGACAATGCAAACC

[4] ------TTTTCAACTTGCATGT-GGAGCCATGTGGCACAAAT-ACTCAT--GCCAGCTCTTTACATTATG-AGCACAGCAG-TTTATTACTCACT

GAAGACAGAATGAA---TGTAG-AAAAGGC-TGAATTCTGTAA-T-A-AAAGCA-AGCAGC-CTGGCTTAG--CAAGG-AGCCAACAGAGCAGAT
GGGCTGAAAGTAAGGAAACA--T-GTAATGATAGGCAGACTCC------CAGCACAG—AGGAAAATGTAGTTCTGAATACTGATCCCCTGAATGG
GAGAAAAG-A-A-CTGAATAAGCAGAA--AC--CTCCATGCTCTGACA-GT-CCTAG-GGATTCCCAAGTTGTTGCATGGATAACACAGAATAGT
AGCATACAGAAAGTTAATGAGTGGTTTTCCA--G----------ACGTGATGAAATATTAACTTCTC-ATAGCTCATGTTAT
GGGAGAGCTGAATCAAATACAGAAGTATCTGGTGCAGTAGAAGTTCCACTTGAAGTAGATGGATTTTCTGGCT-CTACAGAGAAAATAAC-CTTA
ATGACCAGTGATCCTCATGATGCTGTAATA-T--GTGAAAGTGG-AAGAGTCCACTCCAA--ACCATTGGAAAGTACTATTGAAGATA-A-AATA
TTT-GG-GAAAA-C-CTATCGGAGGAAGG-CAAGCCT-CCCTAACTT-CAGCCACA-CA—ACTGAAAACATAATTATAGGAGCATCTGCTGTAGA
ACCTCAGATAACACAAGAG

[5] -CAGG-G-TAGTTCTGTT-TCA-AACTTGCATGTGGAGCCATGTGGCACAAATACTCATGCCAGCTCATTA-CAGCATGAGAACAGCAG-TTTAT
TACTCACTAAAGACAGAATGAATGTAGAA-AAGGCTGAATTCTGTAATAAAAGCGAAC-AGCCTGGCTTAGCAAGGAGCCAACATAACAGATGGG
CTGGAAGTAAGGAAACATGTAATGATAGGCGGACTCCC-A---------G-CAC-----AGAAAAAAAGGTAGATCTGAATGCTGATCCCCTGTG
TGAGAGAAAAGAATGGAATAAGCAGAAACTGCCATG-CTCAGAGAATC-CTAGAGATACT--GAAGATGTT-CCTTGGAT-AACACTA-AATAGC
AGCATT-CAGAAAGTTAATG-AG-TGG-TTTTCCAGA-A----------GTGATGA—ACTGTTAGGTTCTGATGACTCACATGATGGG
GGGTCTGAATC-AAATGCCA-AAG-TAGCTG-ATGTATTGGACGTTCTAAATGAGGTAGATGAATATTC-TGGTTCT-TCAAAGA-AAATAGACT
TACTGGCCAGCG-ATCCTCATGAGG-C-T---T-TAATATGTAAAAG-TGA-AA-GAGTTCACTCCAAATCAGTAGA-GAGTAATA-CTGAAGAC
AAAATATTT---GGGAAAACCTATCGGAGG-AAGGCAA-GCCTCCCCAACTTAAGCCATGTAACTGAAAATCTAATTATAGGAGCATTTGTTACT
GAGCCACAGATAATACAAGAG

[6] GTATTTCTGTTTCAGAC-TTTCACG-TGG-AGCCATGTGGCACAGAT-ACTCATGCCAGC-TCATTACAGCATGAGAAC-AGCAGTTTATTGCTC
-ACTGAAAACAGACTGAATGTAGAAAAGGCTGAATTCTGTAATAAAAGCAA-ACAGCCTGTCTTAGT-AAAGAG-CCAGCA-GAGCAGATGGGCT
GAAAG-T-AA-GGGCACATG-TAAGG-A-TAGGCAG----------ATTCCCAGCACTGAGAAAAAGA-TAGTTCTGAATACTGATCCCCTG-TA
CAGAAGAAAAGAACTGCGTAA-GCAGAAACCTGCATGCCCTGACAGTCCT-GGAGATTCCCAAGATGTTCCTTGGGTAACCCTGAATAATAGC-A
TA--CAGAAAGTT-AATG-ACTGGTTT----------TCCAGAAGTGATGAAATATTAACTTCTGATGACTCGTGCG
ATGGGGGGTCTGAATCAAATAATGA-AGTAGCTGGTG-CAGTGGAAATTC--CAAA-TAAAGTAGATGGATATTCAGGTTCTTCAGAGAAAATCA
ACTTAATGG-CC-AG--TGATCCTCATGGT-ACTTTAATA-CACGAAA-GAGTCC---ACTCCAAA-CCCGTAG-A-GA-GTAATATTG-AAGAT
AAAATATTTGGGAAAACCTATCGGAGG-AAGTCAA-GTCTCCCTAACTTCAGCCACATAGCTGAAGATCTAATTCTAGGCGCATTTACTGTAGAA
CCTCAGATAACACAAGAG

[7] ---CGGGGTGTTTCTTCCTCA-GACTTGAATATGAAGCCATGGAACACAGACATTCATGCTAGCT-CATTACCACCTGA-GATCACCAGTGTATT
GACTAACACA-GTCAGCATGAACATAGAAAAGGCTGAACTCTGTGATAAAAGTAAAAG-GCC-TGGTTTAGCAAGGAGCCAGCAGATAAGTCAG-
-GATAAT-AGTAAGGAAAAATGTAG-TGCT-GGGAAGACCTCA--------TATGCAGA--GGTGCCGCAT-GAGCTGAACCCCCATCATCTGTA
TGAGAGGCAAGAACTAGAGGAACAGCCAGAGTGCCCCAAGTACC-CCAGAGGAAA-TCCTCAAAACTGCTTGTCTGGAACCAAACTGAAAAG-CA
GTATTC---AGAA-AGTTA-A-TGACTGGTTAT-----CCAGA-AGTAATGACATT-----TTAGTCTCTGATTATTCCTCTGTTAGGATCCATG
AACAGAATGCAGAGATGGCTAGTGTCTTAGAAATTGGGCATCCAGA--T-ACCACAGATGGAAATTCTAGCATTTCTGGGA---AGACTGACTTG
GTGGCTGACTCCACTGATG-GTGCC-T-GGCTACATATGTCTGAA---AGAA-GCTGCCC----CAGGCAGGCAGA-GA—ACAACAATATTGAAG

ACAAAATATTTG-GAAAAACCTACCATAGA-AAGTCAG-TTCACACTAATTTGAATTACGTAACTGAAAACTTGATTGTTGGAGCTGTTGCTTCT
GATTGTTTGATCCCTCCAGAG

[8] -----GTGGAG-CCATGTGGCACAGATGCTCGTGCCA-GCTCATTA-CTGC-CTGAC-A-TCACCAGTGTATTGCCTAACACAGACAGCATGAA-
TGTAGAAAAGGCTGA-ACTCTGTGAT--AAAAGC-AAAAGGCCTGATTT-AGCATGGAGCCAGCAGATCAGTCAGGAT-GA-AAGTAAGGAAAAA
TGTATT-GCTGGGAAGACCTC-AGAT-GCAAAGGAGTTACA-TGA-----GCTAAA-TGCCCATCATCTG-TATGAGAGGCAAGAACTAGAAGAG
CAGCC-AGAGTGCCCCAACTACCCCAGAGGAAATTCTCAAAACTGC-TT-GTCTGGAACCAAACTGAA-AA-GTAT-TCAGAAAGTTA-ATGAC-
TGGTTA-TCCAA-AAG--TAATGACAT-TT----------TAGTCTCTGA-TTACTCCTCTGGTAGGATCCAT
GAACAGAATGCAGAGATGGATAGTGTCTTAGAAGA-TGGGCATCCAGATGTTACAGA-TGGAAATT-CTAGCATT-TCTG-GGA-AGA-C-TGAC
TTGG-TGGCTGACTCCACCGATGGTGCCTGGCTACATATG-TCTGA-AAG-AAGCTGCT-C---CAGGAAGGCA-GAGA-GCAACAATATTGAAG
ACAAAATC--TTCGGCAAGACCTACCATAGA-AAATCAG-TTCACATTAATTTGAATTATGTGACTGAAAACTTGATTGTTGGAGCTGTTGATTC
TGATTCTTTGATCCCTCCAGAG

[9] --------TG-TGGCACAGATGC--TCGTGCCACCTCATTACTT-CCTGAAACCACCAGCT-TATCGCC----CA-ACACAGA-CCGAATGAATG
TAGAAAAGGCTG-AACTCTGTAATAAAAGCAAACAGCCTGGCTTAGCAAAAAACCAACAGAGCAG-TCTGGATGA-AAGTAAGGAAAT-ATGTAG
TGCTGGAAAGACCCTGGGTGCCCA-TGAGCTGA-ATGCCCATCA-T--------CCATGCG--AGAGGAAAGAACTAGAGGATGAGCCAC-A-GCA
CCCT-G-AGAGCCCC--A-G-AGGTAATCCTCAG-AACTGCC-AGTCTGG-AACCAAACTGAAAAGTAGTAT-TCAGAAAG-TTAATGAGTGGTT
AT-CCAG-G-AGTAATGATAT-TTTAACT-TCTGATAACTCCTAT----------A-GTAGGAACC-ATGAGCAGAATGCAGAG
ATGCCTA-G-TGCCT-TAAAAGATGGGTATCCAGATACTGCAG-ATGCAAATTCTAACATTTCTGAGAAGACTGACCCAGTGGCTGACATCACTT
ATGATCCCTGGCCACATGTGCCTGAAAGAAGCTGCCCCAGGCC-AGCAG--AAAACAATAA-CATTGAAGATAAAATATTTGGAAAAACCT-ATC
GGAGAAAA-T-CA--GGT-CACCCTCA-TTTGA-ATTGT--AT-AACTGA--A--AACTTGTTTGCTG-G-AGCTGTTGTTGCTC-C-TGATTCT
TTGATC-CC---TCCAGAG---

Fitness: -16023.0

ClustalW Solution Individual:

[1] CAGAGTATTTCTATTTCAAATGTGTGTGTGGAGCCATGTGGCACAGATGCTCATGCCAGCTCATTACAGCCTGAGACCAGCAGTTTATTGCTCAT
TGAAGACAGAATGAATGCAGAAAAGGCTGAATTCTGTAATAAAAGCAAACAGCCTGGCATAGCAGTGAGCCAGCAGAGCAGATGGGCTGCAAGTA
AAGGAACATGTAACGACAGGCAGGTTCCCAGCACTGGGGAAAAGGTAGGTCCAAACGCTGACTCCCTTAGTGATAGAGAGAAGTGGACTCACCCG
CAAAGTCTGTGCCCTGAGAATTCTGGAGCTA----CCACCGATGTT--CCTTGGATAACACTAAATAGCAGCGTTCAGAAAGTTAATGAGTGGTTT
TCCAGAACTGGTGAAATGTTAACTTCTGACAGCGCATCTGCCAGGAGGCACGAGTCAAATGCTGAAGCAGCTGTTGTGTTGGAAGTT------TC
AAACGAAGTGGATGGGGGTTTTAGTTCTTCAAGGAAAACAGACTTAGTAACCCCCGACCCCCATCATACTTTAATGTGTAAAAGTGGAAGAGACT
TCTCCAAACCAGTAGAGG---ATAATATCAGTGATAAAATATTTGGGAAATCCTATCAGAGAAAGGGAAGCCGCCCTCACCTGAACCATGTGACT
GAA------ATTATAGGCACATTTA---TTACAGAACCACAGATAACACAAGAG

[2] CCGCGTATTTCTGTTGCAAACGTGCACGTGGAGCCGTGTGGCACAGATGCTCGTGCCAGCTCATTACAGCGTGGGACCCGCAGTTTATTGTTCAC
TGAGGACAGACTGGATGCAGAAAAGGCTGAATTCTGTGATAGAAGCAAACAGTCTGGCGCAGCAGTGAGCCAGCAGAGCAGATGGGCTGACAGTA
AAGAAACATGTAATGGCAGGCCGGTTCCCCGCACTGAGGGAAAGGCAGATCCAAATGTGGATTCCCTCTGTGGTAGAAAGCAGTGGAATCATCCG
AAAAGCCTGTGCCCTGAGAATTCTGGAGCTA----CCACTGACGTT—CCTTGGATAACACTGAATAGCAGCATTCAGAAAGTGAATGAGTGGTTT
TCCAGAACTGGTGAAATGTTAACTTCTGACAATGCATCTGACAGGAGGCCTGCGTCAAATGCAGAAGCTGCTGTTGTGTTAGAAGTT------TC
AAATGAAGTGGATGGATGTTTCAGTTCTTCAAAGAAAATAGACTTAGTTGCCCCTGATCCCGATAATGCTGTAATGTGTACAAGTGGAAGAGACT
TCTCCAAGCCAGTAGAGA---ATATTATCAACGATAAAATATTTGGGAAAACCTATCAGAGAAAGGGAAGCCGCCCTCACTTGAACCATGTGACT
GAA------ATTATAGGCACATTTA---CTACAGAACCACAGATTATACAAGAG

[3] CAGGGTATTTCTGTTTCAAACTTGCATGTGGAGCCATGTGGCACAAATACTCATGCCAGCTCATTACAGCATGAGAACAGCAGTTTATTACTCAC
TAAACACAGAATGAATGTAGAAAAGGCTGAAATCTGTAATAACAGCAAACAGCCTGGCTTAGCAAGGAGCCAACAGAGCAGATGGGCTGAAAGTA
AGGAAACATGTAATGATAGGCAGATTCCCAGCACAGAGAAAAAGGTAGTTGTGAATGCTGATCTCCTGTGTGGGAGAAAAGAACTGAATAAACAG
AAACCTCCACACTCTGATAGTCCTAGAGATT----CCCAAGATGTT—CCTTGGATAACACTGAATAGTAGCATACGGAAAGTTAATGAGTGGTTT
TCCAGAAGTGACGAAATATTAACTTCTGATGATTCACATGACAGAGGATCTGAATTGAATACTGAAGTAGGTGGTGCAGTAGAAGTT------CC
AAATGAAGTGGGTGAATATTCTGGTTCTTCTGAGAAAATAGACTTAATGGCCAGTGATCCTCAGGATGCTTTCATATGTGAAAGTGAAAGAGTCC
ACACCAAGCCAGTAGGAG---GTAATATCGAAGATAAAATATTTGGAAAAACCTATCGGAGGAAGGCAAGCCTCCCTAAGGTGAGCCACACAACT
GAAGTTCTAACTATAGGAGCGTGTG---CTATAGAACCTCAGACAATGCAAACC

[4] ------------TTTTCAACTTGCATGTGGAGCCATGTGGCACAAATACTCATGCCAGCTCTTTACATTATGAGCACAGCAGTTTATTACTCAC
TGAAGACAGAATGAATGTAGAAAAGGCTGAATTCTGTAATAAAAGCAAGCAGCCTGGCTTAGCAAGGAGCCAACAGAGCAGATGGGCTGAAAGTA
AGGAAACATGTAATGATAGGCAGACTCCCAGCACAGAGGAAAATGTAGTTCTGAATACTGATCCCCTGAATGGGAGAAAAGAACTGAATAAGCAG
AAACCTCCATGCTCTGACAGTCCTAGGGATT----CCCAAGTTGTT—GCATGGATAACACAGAATAGTAGCATACAGAAAGTTAATGAGTGGTTT
TCCAGACGTGATGAAATATTAACTTCTCATAGCTCATGTTATGGGAGAGCTGAATCAAATACAGAAGTATCTGGTGCAGTAGAAGTT------CC
ACTTGAAGTAGATGGATTTTCTGGCTCTACAGAGAAAATAACCTTAATGACCAGTGATCCTCATGATGCTGTAATATGTGAAAGTGGAAGAGTCC
ACTCCAAACCATTGGAAA---GTACTATTGAAGATAAAATATTTGGGAAAACCTATCGGAGGAAGGCAAGCCTCCCTAACTTCAGCCACACAACT
GAAAACATAATTATAGGAGCATCTG---CTGTAGAACCTCAGATAACACAAGAG

[5] CAGGGTAGTTCTGTTTCAAACTTGCATGTGGAGCCATGTGGCACAAATACTCATGCCAGCTCATTACAGCATGAGAACAGCAGTTTATTACTCAC
TAAAGACAGAATGAATGTAGAAAAGGCTGAATTCTGTAATAAAAGCGAACAGCCTGGCTTAGCAAGGAGCCAACATAACAGATGGGCTGGAAGTA
AGGAAACATGTAATGATAGGCGGACTCCCAGCACAGAAAAAAAGGTAGATCTGAATGCTGATCCCCTGTGTGAGAGAAAAGAATGGAATAAGCAG
AAACTGCCATGCTCAGAGAATCCTAGAGATA----CTGAAGATGTT—CCTTGGATAACACTAAATAGCAGCATTCAGAAAGTTAATGAGTGGTTT
TCCAGAAGTGATGAACTGTTAGGTTCTGATGACTCACATGATGGGGGGTCTGAATCAAATGCCAAAGTAGCTGATGTATTGGACGTT------CT
AAATGAGGTAGATGAATATTCTGGTTCTTCAAAGAAAATAGACTTACTGGCCAGCGATCCTCATGAGGCTTTAATATGTAAAAGTGAAAGAGTTC
ACTCCAAATCAGTAGAGA---GTAATACTGAAGACAAAATATTTGGGAAAACCTATCGGAGGAAGGCAAGCCTCCCCAACTTAAGCCATGTAACT
GAAAATCTAATTATAGGAGCATTTG---TTACTGAGCCACAGATAATACAAGAG

[6] CAGGGTATTTCTGTTTCAGACTTTCACGTGGAGCCATGTGGCACAGATACTCATGCCAGCTCATTACAGCATGAGAACAGCAGTTTATTGCTCAC
TGAAAACAGACTGAATGTAGAAAAGGCTGAATTCTGTAATAAAAGCAAACAGCCTGTCTTAGTAAAGAGCCAGCAGAGCAGATGGGCTGAAAGTA
AGGGCACATGTAAGGATAGGCAGATTCCCAGCACTGAGAAAAAGATAGTTCTGAATACTGATCCCCTGTACAGAAGAAAAGAACTGCGTAAGCAG
AAACCTGCATGCCCTGACAGTCCTGGAGATT----CCCAAGATGTT—CCTTGGGTAACCCTGAATAATAGCATACAGAAAGTTAATGACTGGTTT
TCCAGAAGTGATGAAATATTAACTTCTGATGACTCGTGCGATGGGGGGTCTGAATCAAATAATGAAGTAGCTGGTGCAGTGGAAATT------CC
AAATAAAGTAGATGGATATTCAGGTTCTTCAGAGAAAATCAACTTAATGGCCAGTGATCCTCATGGTACTTTAATACACGAA------AGAGTCC
ACTCCAAACCCGTAGAGA---GTAATATTGAAGATAAAATATTTGGGAAAACCTATCGGAGGAAGTCAAGTCTCCCTAACTTCAGCCACATAGCT
GAAGATCTAATTCTAGGCGCATTTA---CTGTAGAACCTCAGATAACACAAGAG

[7] GGGGTGTTTCTTCCTCAGACTTGAATATGAAGCCATGGAACACAGACATTCATGCTAGCTCATTACCACCTGAGATCACCAGTGTATTGACTAAC
ACAGTCAGCATGAACATAGAAAAGGCTGAACTCTGTGATAAAAGTAAAAGGCCTGGTTTAGCAAGGAGCCAGCAGATAAGTCAGGATAATAGTAA
GGAAAAATGTAGTGCTGGGAAGACCTCATATGCAGAGGTGCCGCATGAGCTGAACCCCCATCATCTGTATGAGAGGCAAGAACTAGAGGAACAGC
CAGA---GTGCCCCAAGTACCCCAGAGGAAATCCTCAAAACTGCTTGTCTGGAACCAAACTGAAAAGCAGTATTCAGAAAGTTAATGACTGGTTA
TCCAGAAGTAATGACATTTTAGTCTCTGATTATTCCTCTGTTAGGATCCATGAACAGAATGCAGAGATGGCTAGTGTCTTAGAAATTGGGCATCC
AGATACCACAGATGGAAATTCTAGCATTTCTGGGAAGACTGACTTGGTGGCTGACTCCACTGATGGTGCCTGGCTACATATGTCTGAAAGAAGCT
GCCCCAGGCAGGCAGAGAACAACAATATTGAAGACAAAATATTTGGAAAAACCTACCATAGAAAGTCAGTTCACACTAATTTGAATTACGTAACT
GAAAACTTGATTGTTGGAGCTGTTG---CTTCTGATTGTTTGATCCCTCCAGAG

[8] ----------GTGGAGCCATGTGGCACAGATGCTCGTGCCAGCTCATTACTGCCTGACATCACCAGTGTATTGCCTAAC
ACAGACAGCATGAATGTAGAAAAGGCTGAACTCTGTGATAAAAGCAAAAGGCCTGATTTAGCATGGAGCCAGCAGATCAGTCAGGATGAAAGTAA
GGAAAAATGTATTGCTGGGAAGACCTCAGATGCAAAGGAGTTACATGAGCTAAATGCCCATCATCTGTATGAGAGGCAAGAACTAGAAGAGCAGC
CAGA---GTGCCCCAACTACCCCAGAGGAAATTCTCAAAACTGCTTGTCTGGAACCAAACTGAAA---AGTATTCAGAAAGTTAATGACTGGTTA
TCCAAAAGTAATGACATTTTAGTCTCTGATTACTCCTCTGGTAGGATCCATGAACAGAATGCAGAGATGGATAGTGTCTTAGAAGATGGGCATCC
AGATGTTACAGATGGAAATTCTAGCATTTCTGGGAAGACTGACTTGGTGGCTGACTCCACCGATGGTGCCTGGCTACATATGTCTGAAAGAAGCT
GCTCCAGGAAGGCAGAGAGCAACAATATTGAAGACAAAATCTTCGGCAAGACCTACCATAGAAAATCAGTTCACATTAATTTGAATTATGTGACT
GAAAACTTGATTGTTGGAGCTGTTG---ATTCTGATTCTTTGATCCCTCCAGAG

[9] ----------TGTGGCACAGATGCTCGTGCCACCTCATTACTTCCTGAAACCACCAGCTTATCGCCCAA
CACAGACCGAATGAATGTAGAAAAGGCTGAACTCTGTAATAAAAGCAAACAGCCTGGCTTAGCAAAAAACCAACAGAGCAGTCTGGATGAAAGTA
AGGAAATATGTAGTGCTGGAAAGACCCTGGGTGCC---------CATGAGCTGAATGCCCATCATCCATGCGAGAGGAAAGAACTAGAGGATGAG
CCACA---GCACCCTGAGAGCCCCAGAGGTAATCCTCAGAACTGCCAGTCTGGAACCAAACTGAAAAGTAGTATTCAGAAAGTTAATGAGTGGTT
ATCCAGGAGTAATGATATTTTAACTTCTGATAACTCCTATAGTAGGAACCATGAGCAGAATGCAGAGATGCCTAGTGCCTTAAAAGATGGGTATC
CAGATACTGCAGATGCAAATTCTAACATTTCTGAGAAGACTGACCCAGTGGCTGACATCACTTATGATCCCTGGCCACATGTGCCTGAAAGAAGC
TGCCCCAGGCCAGCAGAAAACAATAACATTGAAGATAAAATATTTGGAAAAACCTATCGGAGAAAATCAGGTCACCCTCATTTGAATTGTATAAC
TGAAAACTTGTTTGCTGGAGCTGTTGTTGCTCCTGATTCTTTGATCCCTCCAGAG

```
Fitness: 9354.0


Statistics:
Time Elapsed (HH:mm:ss:SS): 00:04:36:726
Population Size: 400
Number of Unchanged Rounds: 50
Number of Total Rounds: 200
Goodbye, and thank you for flying GAMSA.
******
```

DNA BRCA1 Sequences – No Maximum Number of Rounds Constraint

```
******
GAMSA Best Individual:

[1]   CAGAGTATTTCTATT-TCAAATGTGTGTGTGGAGCCATGTGGCACAGATGCTCATGCCAGCTCATTACAGCCTGAGACCAGCAGTTTATTGCTCA
      TTGAAGACAGA-----ATGAATGCAGAAAAGGCTGAATTCTGTAA-TAAAAGCAAACAGCCTGGCATAGCAGTGAGCCAGCAGAGCAGATGGGCT
      GCAAGTAAAGGAACATGTAACGACAGGCAGGTTCCCAGCACTGGGGAAAAGGTAGGTCCAAAC-GCTG-ACTCCCTTA-G----TGATAGAG---
      AGAAGTGGACTCACCCGCA-AAGTCTG-TGCCCT-GAGAATTCTGGAGC-TACCACCGATGTTCCTTGGATAACACTAAATAGCA-GCGTTCAGA
      -AAGTTAATGAGTGGTTTTCCAGAACTGGTGAAATGTT-AACTTCTGACAGCGCATCTGCCAGGAGGCACGAGTCAAATGCTGAAG-CAGCTGTT
      GTGTTGGAAGTTTCAAACG-AAGTGGATGGGGGTTTTAGTTCTTCAAGGAAA-ACAGAC-TTAGTAACCCCCGACCCCCATC-ATACTTT-AATG
      TGTAAAAGTGGAAGAGACTTCTC-CAAACCAG----TAGAGGATAATATCAG-TGATA-AAA--TATTTG-G-GAAATCCTATCAGAGAA-AGG-
      ---GAAGCC-------GCCCTCACCTGAACCATGTGACTGAAATT-A----------TAGGCACATTTATTACAGAACCACAGATAACACAAGAG

[2]   CCGCGTATTTCTG-T-TGCAAACGTGCACGTGGAGCCGTGTGGCACAGATGCTCGTGCCAGCTCATTACAGCGTGGGACCCGCAGTTTATTGTTC
      ACTGAGGACAGACTGGATGCAGAAAAGGCTGAATTCTGTGAT-AGA-AGCAAACAGTCTGGCGCAGCAGTGAGCCA-GCAGAGCAGATGGGCTGA
      CAGTAA-AG-AA-ACATGTAATGGCAGGCCGGTTCCCCGC---ACTGAGGGAAA---GGCAGAT-C-CAAATGTGG---ATTCCCTCTGTG-GTA
      -G-AAA-G-CAGTGGAATCATCCGAAAAGCCTGTGCCCTG--AGAATTCTGGAGCTACCACTGACGTTCC-TTG----GATAACACTGAA-TAGC
      -AGCA--TTCAGAAAGTGAATGAGTGGTTT-TCCAGAAC-TGGTGAAATGTTAACTTCTGACAATGCATCTGACAGGAGGCCT-GCGTCAAATGC
      AGAAGCTGCTGTTGTGTTAG-AAGTTTCAAATG-AAGTGGATG-GATGTTTCAGTTCTTCA-AAGAAAATAGACTTAGTTGCC---CCTGATCCC
      -GATAATGCTGTAATGTGTACAAGTGGAAGAGACTTCTCCAAGCCAGTAGAGAATATTA-TCA-ACGAT----AAAATATTTG-G-GAAAACCTA
      TCAGAGAA----AGGGAAGCCGCCC-TCACTTGAACCATGTGACTGAAATT---AT-AG-GC-ACATTTACTACAGAACCACAGATTATACAAGA
      G

[3]   CAGGGTATTTCTGTT-TCAAACTTGCATGTGGAGCCATGTGGCACAAATACTCATGCCAGCTCATTACAGCATGAGAACAGCAGTTTATTACTCA
      CTAAACACA-----GAATGAATGTAGAAAAGGCTGAAATCTGTAA-TAACAGCAAACAGCCTGGCTTAGCAAGGAGCCAACAGAGCAGATGGGCT
      GAAAGTAAGGAAACATGTAATGATAGGCAGATTCCCAGCACAGA----GA-AAAAGGTAGTTGTGAATGCTGATCT------CCTGTG-TGGGAG
      AAAA-GAACTGAATAA-AC-AGAAACCTCCACACTCTGATAGTC--C--TAGAGATTCC-CAAGATGTTCCT-----TGGATAA-CACTGAATAG
      TAGCATACGGAAA-GTTA--ATGAGTGGTT-TTCCAGA-AGTGACGAAATATTAACTTCTGATGATTCACA-TGACAGAG-GATCTGAATTGAAT
      ACTGAAGTAGGTGGTGCAG-TAGAAGTTCCAAATGAAGTGGGTGAATATTCTGGTTCTTCTGAGAAAATAGACTTAATGG--CC-AGTGATC-CT
      CAGGATGCTT-T-CATATGTGAAAGTGAAAG--A-GTCCACA-CCAAGCCAGTAGGAGGTAATAT-CGAA-GATAAAATATTTGGAAAAACCTAT
      CGGAG-G--A-AGGCAAGCCTCCCTA-AGGTGAGCCACACAACTGAAGTTCTAACTATAGGAGCGTGTGCTATAGAACCTCAGACAATGCAAACC

[4]   ----------T-TTTCAACTTGCATGTGGAGCCATGTGGCACAAATACTCATGCCAGCTCTTTACATTATGAGCACAGCAGTTTATTACTCACT-
      --G-AAGACAGAA-TGAATGTAGAAAAGGCTGAATTCTGTAA-TAAAAGCAAGCAGCCTGGCTTAGCAAGGAGCCAACAGAGCAGATGGGCTGAA
```

AGTAAGGAAACATGTAATGATAGGCAGACTCCCAGCACAG------AGGAAAAT-GTAGT-TCTGAATACTG-----ATCCCCTGAATGGG-AGA
AAAGAACTGA--ATAAGCAGAAACCTCCATGCTCTGACAGTCCTAGGG-AT--T-CC-CAAGTTG-TTGC-A-T-GGATAAC—ACAGAATAGTAG
CATACAGAAAGTTAATGAGTGG-TTTTCCAGACG-TGATGAAATA-TTA-ACTTCTCATAGCTCATGTTATGGGAGAGCTGAATCAAATACAGAA
GTATCTGGTGCAGTAG-AAGTTCCACTTGAAGT-AGATGGATTTTCTGGCTCTACAGAGAAAATAACCTTAA---TG--ACCAGTGATCCTC-AT
GATGCTGTAATA--TGTGAAAGTGGAAGAGTCCAC--TC-CAAACCATTGGAA-A-GTACTATTGAA-GATAAAATATTTGGGAAAACCTATCGG
AG-G--A-AGGCAAGCCTCCCTA-ACTTCAGCCACACAACTGAAAACATAATTATAGGAGCATCTGCTGTAGAACCTCAGATAACACAAGAG

[5] CAGGGTAGTTCTGTT—TCAAACTTGCATGTGGAGCCATGTGGCACAAATACTCATGCCAGCTCATTACAGCATGAGAACAGCAGTTTATTACTCA
CTAAAGACAGA-----ATGAATGTAGAAAAGGCTGAATTCTGTAA-TAAAAGCGAACAGCCTGGCTTAGCAAGGAGCCAACATAACAGATGGGCT
GGAAGTAAGGAAACATGTAATGATAGGCGGACTCCCAGCA--CAGAA--AAAA-AGGTA-G---ATCTGAA-TGCT---GATCCCCTGTGTGA--
-GAGAAAAGAATGGAATAAGCAGAAACTGCCATGCTCAGAG-AA-TCCTAGAGATACTGAAGATGTTCCTTGGA-----TA-ACACTAAATAGCA
GCATT--CAGAAAG-TTAATGAGTGGT-TTTCCAGAAG-TGATGAACTGTTAGGTTCTGATGACTCACATGATGGGGGGTCTGAAT—CAAATGCC
AAAGTAGCTGATGTATTGG-ACGTTCTAAATG---AGGTAGATGAATATTCTGGTTCTTCAAAGAAAATAGACTTACTGGCCAGCGATC--C-TC
ATGAG-GCTTTAATATGTAAAAGTGAAAGAGTTCACTCCAAATCAGTAG--AGAGTAA--TACTGAA-GAC-AAAATATTTGGG—AAAACCTATC
GGAGGA----AGGCAAGCCTCCCCA-ACTTAAGCCATGTAACTGAAAATCTAATTATAGGAGCATTTGTTACTGAGCCACAGATAATACAAGAG

[6] AGGGTATTTCTGTTTCAGACTTTCACGTGGAGCCATGTGGCACAGA-TAC-TCATGCCAGCTCATTACAGCATGAGAACAGCAGTTTAT-TG-CT
CACT-GAAAACA-GACTGAA-TGTAGAAAAGGCTGAATTCTGTAA-TAAAAGCAAACAGCCTGTCTTAGTAAAGAGCCAGCAGAGCAGATGGGCT
GAAAGTAAGGGCACATGTAAGGATAGGCAGATTCCCAGCACTGAGAAAAAGA-----TAGTTC-T-GAATACTGAT-----CCCCTGTACAGAA-
GAAAAGAACTGCGTAAGCAGAA--A-CCTGCATGCCCTGACAG-T-CCTGGAG--ATT-CCCAAGATGTTCCT-----T-GGGTAACCCTGAATA
ATAGCATACAGAAAGTTAATGACTGGT-TTTCCAGAAGTGA-TGAAATATTAACTTCTGATGACTCGTGCGATGGGGGGTCTGAATCAAA—TAAT
GAAGTAGCTGGTGCAGTGG-AAATTCCAAATAAAGT-AGATGGATA--TTCAGGTTCTTCAGAGAAAATCAACTTAATGGCCAGTGATCCTCAT-
GGTAC--------TTTA-ATACACGAAAGAGTC--CA--CTC-CAAA-CCCGTAGAGAGTAATATT-GAA-GATAAAATATTTGGGAAAACCTAT
CGGAG-G--A-AGTCAAGTCTCCCTA-ACTTCAGCCACATAGCTGAAGATCTAATTCTAGGCGCATTTACTGTAGAACCTCAGATAACACAAGAG

[7] GGGGTG-T-TTCTTCCTCAGACTTGAATATGAAGCCATGGAACACAGACATTCATGCTAGCTCATTACCACCTGAGATCACCAGTGTATTGACTA
ACACAGTCAGCATGAACATAGAAAAGGCTGAACTCTGTGATAAAAGTAAAAGGC--CTGGTTTAGCAAGGAGCCA-GCAGATAAG--TCA-GGAT
AATAGTAA-GG-A-AAAA-TGTAGTGCTGGGAAGACCTCA--TATGCAGAGGTGCCGCATGAG--CTGAACCC---C------CATCATCTGTA-
TGAGAGG-CAAGAACTAGAGGAACAGCCAGAGTGCCCCAAGTACCCCAG-AGGAAATCCTCAAAACTGCTTGTC----TGGAACCAAACTGAAAA
GCAGTATTCAGAAAGTTAATGACTGGT-TATCCAGAAG-TAATGACATTTTAGTCTCTGATTATTCCTCTGTTAGGATCCATGAACAGAATGCAG
AGATGGCTAGTGT-CTTAGA-AATTGGGCATCCAGATACCACAGATG—GAAATTCTAGCATTTCTGGGAAGACTGACTTGGTGGCTGACTCCACT
GATGGTGCCTGGCTACATATGTCTGAAAGAAGCTGCCCCAGG-CAGGCAGA-G-AACAACAATAT-TGAA-GACAAAATATTTGGAAAAACCTAC
CAT-AGA--A-AGTCAGTTCACAC-TAATTTGAATTACGTAACTGAAAACTTGATTGTTGGAGCTGTTGCTTCTGATTGTTTGATCCCTCCAGAG

[8] ----GTGGAGCCATGTGGCACAGATGCTCGTGCCAGCTCATTACTGCCTGACA-TCACCAGTGTATTGCCTAACACAGACAGCATGAATGTAGAA
AAGGCTGAA-CTCTGTGATAAAAGC-AAAAGGCCTGATTTAGCATGGAGCCAGCAG-ATCAGTCAGGATGAAAGTAAGGAAAAA--TGTAT-TGC

TGGGAAGACCTC-AGATGC-AAAGGAGTTACATG---AGCT-AAATGCCCATCATCTGTATGAG-AGGCAAGAAC---------TAGAAGAGCAG
-CCAGAGTGCCCCAA--CTA-CCCCAGAGGAAATTCTCAAAACTGCTTGTCTGGAACCAAACTGAAAAGTAT-------TCAGAA---AGTTAAT
GACTGGTTATCCAAAAGTAATGACATTTTAGTCTCTGAT-TACTCCTCTGGTAGGA-TCCATGAACAGAATGCAG-AG--ATGGA----TAGTGT
CTT-AGAAGATGGGC--ATCC-A-GATGTTACAGATGGAAATTCTAGCATTTCTGGGAAGACTGACTTGGTGG-CTGAC-TCC--AC—CGATGGT
GCCTGG----C-TACA--TATGTCTGAAAGAAGCTGCTCCAGGAAGGCA-GAGAGCAACAATAT-TG--AA-GACAAAATCTTCGGCAAGACCTA
CCATAGAA----AATCAGTTCAC-ATTAATTTGAATTATGTGACTGAAAACTTGATTGTTGGAGCTGTTGATTCTGATTCTTTGATCCCTCCAGA
G

[9] ---TGTGGCACAGATG---CTCGTGCCACCTCATTACTTCCTGAAACCACCAGCTT-ATCGCCCAACACAGACCGAATGA---A-TGTAGAAAAG
GCTGAACTCTG----TAATAAAAGCAAACAGCCTGGCTTAGCAAAAAACCAACAGAGCAGTCTGG-A-TGAAAGTAAGGAA-ATAT-G-TAGTGC
TGGAAAGACCCTGGGTGCCCATGAGCTGAATGC--CCATCATCCATGCGAGAGGAAAG-AACTA-G-AGGATGAGCC--------ACAGC-ACCC
-TGAGAGCCCCAGAGGT-AATCCTCAGAACTGCCAGTCTGGAACCAAACT-GAAAAGTAGTATTCAGAAAGTTA-----ATGAGTG-GTTATCCA
-GGAGTAAT--GATATT--TTAACTTCTGATAACTCC---TA-TAGTAGG-AACCATGAGCAGAAT-G-CAGA--GATGCCTAGTGCCTTA-AAA
GATGGGTATCCAGA-TACT-GCA-GATGCAAATTCTAACATT----TCTGAGAAGACTGAC-CCAGTG-GCTGACATC-ACT-----TATGATCC
CTGGCCAC----ATGTGCC--TGAAAG-AAGCTGCCCCAGGCCAGCAG----AAAACAATAACATTGAAGAT-AAAATA-TTT—GGAAAAACCTA
TCGGAGAAAATCAGGTCACCCTCATTTGAATTGTATAACTGAAAACTTGTTT-GCTGG-AGCTGTTGTTGCTCCTGATTCTTTGATCCCTCCAGA
G

Fitness: -6986.0

ClustalW Solution Individual:

[1] CAGAGTATTTCTATTTCAAATGTGTGTGTGGAGCCATGTGGCACAGATGCTCATGCCAGCTCATTACAGCCTGAGACCAGCAGTTTATTGCTCAT
TGAAGACAGAATGAATGCAGAAAAGGCTGAATTCTGTAATAAAAGCAAACAGCCTGGCATAGCAGTGAGCCAGCAGAGCAGATGGGCTGCAAGTA
AAGGAACATGTAACGACAGGCAGGTTCCCAGCACTGGGGAAAAGGTAGGTCCAAACGCTGACTCCCTTAGTGATAGAGAGAAGTGGACTCACCCG
CAAAGTCTGTGCCCTGAGAATTCTGGAGCTA----CCACCGATGTT—CCTTGGATAACACTAAATAGCAGCGTTCAGAAAGTTAATGAGTGGTTT
TCCAGAACTGGTGAAATGTTAACTTCTGACAGCGCATCTGCCAGGAGGCACGAGTCAAATGCTGAAGCAGCTGTTGTGTTGGAAGTT------TC
AAACGAAGTGGATGGGGGTTTTAGTTCTTCAAGGAAAACAGACTTAGTAACCCCCGACCCCCATCATACTTTAATGTGTAAAAGTGGAAGAGACT
TCTCCAAACCAGTAGAGG---ATAATATCAGTGATAAAATATTTGGGAAATCCTATCAGAGAAAGGGAAGCCGCCCTCACCTGAACCATGTGACT
GAA------ATTATAGGCACATTTA---TTACAGAACCACAGATAACACAAGAG

[2] CCGCGTATTTCTGTTGCAAACGTGCACGTGGAGCCGTGTGGCACAGATGCTCGTGCCAGCTCATTACAGCGTGGGACCCGCAGTTTATTGTTCAC
TGAGGACAGACTGGATGCAGAAAAGGCTGAATTCTGTGATAGAAGCAAACAGTCTGGCGCAGCAGTGAGCCAGCAGAGCAGATGGGCTGACAGTA
AAGAAACATGTAATGGCAGGCCGGTTCCCCGCACTGAGGGAAAGGCAGATCCAAATGTGGATTCCCTCTGTGGTAGAAAGCAGTGGAATCATCCG
AAAAGCCTGTGCCCTGAGAATTCTGGAGCTA----CCACTGACGTT—CCTTGGATAACACTGAATAGCAGCATTCAGAAAGTGAATGAGTGGTTT
TCCAGAACTGGTGAAATGTTAACTTCTGACAATGCATCTGACAGGAGGCCTGCGTCAAATGCAGAAGCTGCTGTTGTGTTAGAAGTT------TC

AAATGAAGTGGATGGATGTTTCAGTTCTTCAAAGAAAATAGACTTAGTTGCCCCTGATCCCGATAATGCTGTAATGTGTACAAGTGGAAGAGACT
TCTCCAAGCCAGTAGAGA---ATATTATCAACGATAAAATATTTGGGAAAACCTATCAGAGAAAGGGAAGCCGCCCTCACTTGAACCATGTGACT
GAA------ATTATAGGCACATTTA---CTACAGAACCACAGATTATACAAGAG

[3] CAGGGTATTTCTGTTTCAAACTTGCATGTGGAGCCATGTGGCACAAATACTCATGCCAGCTCATTACAGCATGAGAACAGCAGTTTATTACTCAC
TAAACACAGAATGAATGTAGAAAAGGCTGAAATCTGTAATAACAGCAAACAGCCTGGCTTAGCAAGGAGCCAACAGAGCAGATGGGCTGAAAGTA
AGGAAACATGTAATGATAGGCAGATTCCCAGCACAGAGAAAAAGGTAGTTGTGAATGCTGATCTCCTGTGTGGGAGAAAAGAACTGAATAAACAG
AAACCTCCACACTCTGATAGTCCTAGAGATT----CCCAAGATGTT--
CCTTGGATAACACTGAATAGTAGCATACGGAAAGTTAATGAGTGGTTTTCCAGAAGTGACGAAATATTAACTTCTGATGATTCACATGACAGAGG
ATCTGAATTGAATACTGAAGTAGGTGGTGCAGTAGAAGTT------CCAAATGAAGTGGGTGAATATTCTGGTTCTTCTGAGAAAATAGACTTAA
TGGCCAGTGATCCTCAGGATGCTTTCATATGTGAAAGTGAAAGAGTCCACACCAAGCCAGTAGGAG---GTAATATCGAAGATAAAATATTTGGA
AAAACCTATCGGAGGAAGGCAAGCCTCCCTAAGGTGAGCCACACAACTGAAGTTCTAACTATAGGAGCGTGTG---CTATAGAACCTCAGACAAT
GCAAACC

[4] ----------TTTTCAACTTGCATGTGGAGCCATGTGGCACAAATACTCATGCCAGCTCTTTACATTATGAGCACAGCAGTTTATTACTCAC
TGAAGACAGAATGAATGTAGAAAAGGCTGAATTCTGTAATAAAAGCAAGCAGCCTGGCTTAGCAAGGAGCCAACAGAGCAGATGGGCTGAAAGTA
AGGAAACATGTAATGATAGGCAGACTCCCAGCACAGAGGAAAATGTAGTTCTGAATACTGATCCCCTGAATGGGAGAAAAGAACTGAATAAGCAG
AAACCTCCATGCTCTGACAGTCCTAGGGATT----CCCAAGTTGTT—GCATGGATAACACAGAATAGTAGCATACAGAAAGTTAATGAGTGGTTT
TCCAGACGTGATGAAATATTAACTTCTCATAGCTCATGTTATGGGAGAGCTGAATCAAATACAGAAGTATCTGGTGCAGTAGAAGTT------CC
ACTTGAAGTAGATGGATTTTCTGGCTCTACAGAGAAAATAACCTTAATGACCAGTGATCCTCATGATGCTGTAATATGTGAAAGTGGAAGAGTCC
ACTCCAAACCATTGGAAA---GTACTATTGAAGATAAAATATTTGGGAAAACCTATCGGAGGAAGGCAAGCCTCCCTAACTTCAGCCACACAACT
GAAAACATAATTATAGGAGCATCTG---CTGTAGAACCTCAGATAACACAAGAG

[5] CAGGGTAGTTCTGTTTCAAACTTGCATGTGGAGCCATGTGGCACAAATACTCATGCCAGCTCATTACAGCATGAGAACAGCAGTTTATTACTCAC
TAAAGACAGAATGAATGTAGAAAAGGCTGAATTCTGTAATAAAAGCGAACAGCCTGGCTTAGCAAGGAGCCAACATAACAGATGGGCTGGAAGTA
AGGAAACATGTAATGATAGGCGGACTCCCAGCACAGAAAAAAAGGTAGATCTGAATGCTGATCCCCTGTGTGAGAGAAAAGAATGGAATAAGCAG
AAACTGCCATGCTCAGAGAATCCTAGAGATA----CTGAAGATGTT—CCTTGGATAACACTAAATAGCAGCATTCAGAAAGTTAATGAGTGGTTT
TCCAGAAGTGATGAACTGTTAGGTTCTGATGACTCACATGATGGGGGGTCTGAATCAAATGCCAAAGTAGCTGATGTATTGGACGTT------CT
AAATGAGGTAGATGAATATTCTGGTTCTTCAAAGAAAATAGACTTACTGGCCAGCGATCCTCATGAGGCTTTAATATGTAAAAGTGAAAGAGTTC
ACTCCAAATCAGTAGAGA---GTAATACTGAAGACAAAATATTTGGGAAAACCTATCGGAGGAAGGCAAGCCTCCCCAACTTAAGCCATGTAACT
GAAAATCTAATTATAGGAGCATTTG---TTACTGAGCCACAGATAATACAAGAG

[6] CAGGGTATTTCTGTTTCAGACTTTCACGTGGAGCCATGTGGCACAGATACTCATGCCAGCTCATTACAGCATGAGAACAGCAGTTTATTGCTCAC

TGAAAACAGACTGAATGTAGAAAAGGCTGAATTCTGTAATAAAAGCAAACAGCCTGTCTTAGTAAAGAGCCAGCAGAGCAGATGGGCTGAAAGTA
AGGGCACATGTAAGGATAGGCAGATTCCCAGCACTGAGAAAAAGATAGTTCTGAATACTGATCCCCTGTACAGAAGAAAAGAACTGCGTAAGCAG
AAACCTGCATGCCCTGACAGTCCTGGAGATT----CCCAAGATGTT--CCTTGGGTAACCCTGAATAATAGCATACAGAAAGTTAATGACTGGTTT
TCCAGAAGTGATGAAATATTAACTTCTGATGACTCGTGCGATGGGGGGTCTGAATCAAATAATGAAGTAGCTGGTGCAGTGGAAATT------CC
AAATAAAGTAGATGGATATTCAGGTTCTTCAGAGAAAATCAACTTAATGGCCAGTGATCCTCATGGTACTTTAATACACGAA------AGAGTCC
ACTCCAAACCCGTAGAGA---GTAATATTGAAGATAAAATATTTGGGAAAACCTATCGGAGGAAGTCAAGTCTCCCTAACTTCAGCCACATAGCT
GAAGATCTAATTCTAGGCGCATTTA---CTGTAGAACCTCAGATAACACAAGAG

[7] CGGGGTGTTTCTTCCTCAGACTTGAATATGAAGCCATGGAACACAGACATTCATGCTAGCTCATTACCACCTGAGATCACCAGTGTATTGACTAA
CACAGTCAGCATGAACATAGAAAAGGCTGAACTCTGTGATAAAAGTAAAAGGCCTGGTTTAGCAAGGAGCCAGCAGATAAGTCAGGATAATAGTA
AGGAAAAATGTAGTGCTGGGAAGACCTCATATGCAGAGGTGCCGCATGAGCTGAACCCCCATCATCTGTATGAGAGGCAAGAACTAGAGGAACAG
CCAGA---GTGCCCCAAGTACCCCAGAGGAAATCCTCAAAACTGCTTGTCTGGAACCAAACTGAAAAGCAGTATTCAGAAAGTTAATGACTGGTT
ATCCAGAAGTAATGACATTTTAGTCTCTGATTATTCCTCTGTTAGGATCCATGAACAGAATGCAGAGATGGCTAGTGTCTTAGAAATTGGGCATC
CAGATACCACAGATGGAAATTCTAGCATTTCTGGGAAGACTGACTTGGTGGCTGACTCCACTGATGGTGCCTGGCTACATATGTCTGAAAGAAGC
TGCCCCAGGCAGGCAGAGAACAACAATATTGAAGACAAAATATTTGGAAAAACCTACCATAGAAAGTCAGTTCACACTAATTTGAATTACGTAAC
TGAAAACTTGATTGTTGGAGCTGTTG---CTTCTGATTGTTTGATCCCTCCAGAG

[8] ----------GTGGAGCCATGTGGCACAGATGCTCGTGCCAGCTCATTACTGCCTGACATCACCAGTGTATTGCCTAA
CACAGACAGCATGAATGTAGAAAAGGCTGAACTCTGTGATAAAAGCAAAAGGCCTGATTTAGCATGGAGCCAGCAGATCAGTCAGGATGAAAGTA
AGGAAAAATGTATTGCTGGGAAGACCTCAGATGCAAAGGAGTTACATGAGCTAAATGCCCATCATCTGTATGAGAGGCAAGAACTAGAAGAGCAG
CCAGA---GTGCCCCAACTACCCCAGAGGAAATTCTCAAAACTGCTTGTCTGGAACCAAACTGAAA---AGTATTCAGAAAGTTAATGACTGGTT
ATCCAAAAGTAATGACATTTTAGTCTCTGATTACTCCTCTGGTAGGATCCATGAACAGAATGCAGAGATGGATAGTGTCTTAGAAGATGGGCATC
CAGATGTTACAGATGGAAATTCTAGCATTTCTGGGAAGACTGACTTGGTGGCTGACTCCACCGATGGTGCCTGGCTACATATGTCTGAAAGAAGC
TGCTCCAGGAAGGCAGAGAGCAACAATATTGAAGACAAAATCTTCGGCAAGACCTACCATAGAAAATCAGTTCACATTAATTTGAATTATGTGAC
TGAAAACTTGATTGTTGGAGCTGTTG---ATTCTGATTCTTTGATCCCTCCAGAG

[9] ----------TGTGGCACAGATGCTCGTGCCACCTCATTACTTCCTGAAACCACCAGCTTATCGCCCAA
CACAGACCGAATGAATGTAGAAAAGGCTGAACTCTGTAATAAAAGCAAACAGCCTGGCTTAGCAAAAAACCAACAGAGCAGTCTGGATGAAAGTA
AGGAAATATGTAGTGCTGGAAAGACCCTGGGTGCC---------CATGAGCTGAATGCCCATCATCCATGCGAGAGGAAAGAACTAGAGGATGAG
CCACA---GCACCCTGAGAGCCCCAGAGGTAATCCTCAGAACTGCCAGTCTGGAACCAAACTGAAAAGTAGTATTCAGAAAGTTAATGAGTGGTT
ATCCAGGAGTAATGATATTTTAACTTCTGATAACTCCTATAGTAGGAACCATGAGCAGAATGCAGAGATGCCTAGTGCCTTAAAAGATGGGTATC
CAGATACTGCAGATGCAAATTCTAACATTTCTGAGAAGACTGACCCAGTGGCTGACATCACTTATGATCCCTGGCCACATGTGCCTGAAAGAAGC
TGCCCCAGGCCAGCAGAAAACAATAACATTGAAGATAAAATATTTGGAAAAACCTATCGGAGAAAATCAGGTCACCCTCATTTGAATTGTATAAC
TGAAAACTTGTTTGCTGGAGCTGTTGTTGCTCCTGATTCTTTGATCCCTCCAGAG

Fitness: 9354.0

```
Statistics:
Time Elapsed (HH:mm:ss:SS): 01:10:43:992
Population Size: 400
Number of Unchanged Rounds: 50
Number of Total Rounds: 2785
Goodbye, and thank you for flying GAMSA.
******
```