Genetic Algorithms Applied to Computational Genomics

Sanders Chong*
Computer Science Department
Stanford University
(Dated: June 4, 2002)

This paper uses genetic algorithms as a heuristic for analyzing DNA/RNA sequences. In an effort to improve the performance of current gene analyzing tools, genetic algorithms have been employed to reduce both space and time complexity of finding solutions to the most fundamental operations in computational genomics. In this paper, the central problems considered are gene alignments (global and local) and motif finding. This paper formulates sample solutions to these problems and analyzes their performance. The methods described here can be extended to solve problems in multiple sequence alignments and gene prediction. Furthermore, the focus of this paper is geared more toward a discssion of computer science and genetic algorithms than topics in biology.

INTRODUCTION AND OVERVIEW

In recent years, the tremendous push to sequence the entire human genome has left us with a wealth of information about human genetics. However, this information is essentially in an encrypted form. The task in upcoming years will be to analyze these sequences to decrypt the inner workings of the human body. Without the proper analysis tools, our understanding of genetics will be minimal.

Among the most fundamental methods employed for gene analysis are sequence alignment and motif finding.

Sequence alignment is essential in comparing gene sequences. It offers insight into how genes differ and which nucleotides give rise to what functionality. Similarly, searching for common motifs in multiple sequences allows geneticists to uncover similarities or differences between genes. Conventional methods to analyze these gene sequences are still in their incipient stages. In analyzing these sequences, there is often a trade-off between the time to search for an answer and the quality of results produced.

For the most accurate gene analysis, researchers must apply methods that are often greater than quadratic in space and time with respect to the length of the sequences they are analyzing. This means even for sequences a few thousand base pairs long the space and time complexity becomes overwhelming for most practical purposes.

Recently, researchers have shifted toward probabilistic models that give "close-to-optimal" results in far less space and time. This shift has led to great progress in gene analysis, but there is still a lot of interest in making these heuristics more accurate or more efficient.

Genetic algorithms are good candidates for such heuristics because genetic algorithms are sensitive and selective in searching for optimal results. The fitness measure associated with sequence alignment and motif finding, too, are well suited for genetic algorithms.

Background and Statement of Problem:

To understand the methods presented later in this paper, it is important that the reader understands the algorithms presented in this section. If you are familiar with the dynamic programming approach to sequence alignment and Gibbs Sampling for motif finding, feel free to skip this section.

Global Alignment:

The best method to deterministically find global alignments is given by a dynamic programming approach. This method, however, has a space and time complexity of at least nm, where n and m are the lengths of 2 sequences being compared. For large sequences (even a few kilo-base pairs), this space and time requirement becomes too large for practical purposes, especially for multiple sequences.

The genetic algorithm approach will use a representation similar to that of the dynamic programming approach. However, it will only try a subset of valid paths that will evolve to the "optimal" path. To understand how the genetic algorithm will work, it is important to first understand the dynamic programming approach.

The dynamic programming matrix (for a convex gap score) is given by the following:

Given: 2 Sequences, X and Y

Matrix Dimensions: length of X by length of Y ($|X|$ by $|Y|$)

Initialization:

```
A(0,0) = 0
for i = 1 to |X|-1
   A(0,i) = A(0,i-1) - Gap(i)
   A(i,0) = A(i-1,0) - Gap(i)
```

Where Gap(i) is the gap score for the i^{th} consecutive gap

Recursion:

$$A(i,j) = max \left\{ \begin{array}{c} A(i-1,j-1) + s(X_i, Y_j), \\ A(k,j) - Gap(i-k), k = [0, i-1] \\ A(i,k) - Gap(j-k), k = [0, j-1] \end{array} \right\}$$

Given the above model, we can see that each cell depends only on 3 previously calculated cells. Thus, under a constant gap penalty ($\forall$ i Gap(i) = c for c $\in \Re$) there is no need to vary k. Each iteration to find A(i,j), then, can be done in constant time. The total run-time to find an optimal path would be $O(n^2)$, where n is the maximum of $|X|$ and $|Y|$. However, given a convex gap penalty, Gap(i) varies over a number of possible values proportional to n, increasing the run-time to find the optimal path to $O(n^3)$.

The space requirement for the algorithm is quadratic with respect to n. With some clever programming, the affine and constant gap penalty model can be reduced to linear space, but given an arbitrary gap function, the space requirement remains quadratic.

As we increase the complexity of the gap model to reflect actual gap penalties in biology, the space and time requirements grow too. The model above, for instance, does not permit consecutive gaps between sequences. To account for this case and allow arbitrary gap functions, we must add two matrices to keep track of optimal gapping penalty scores. This gives us:

$$G(i,j) = max \left(\begin{array}{c} A(k,j) - Gap_y(i-k), k = [0, j-1] \\ H(k,j) - Gap_y(i-k), k = [0, j-1] \end{array} \right)$$

$$H(i,j) = max \left(\begin{array}{c} A(i,k) - Gap_x(j-k), k = [0, i-1] \\ G(i,k) - Gap_x(j-k), k = [0, i-1] \end{array} \right)$$

$$A(i,j) = max \left(\begin{array}{c} A(i-1,j-1) + s(X_i, Y_j), \\ G(i-1,j-1) + s(X_i, Y_j), \\ H(i-1,j-1) + s(X_i, Y_j), \\ G(i,j), \\ H(i,j) \end{array} \right)$$

The genetic algorithm approach will perform a global alignment in linear (large constant) or quadratic time with a linear space requirement.

Local Alignment:

There are a number of heuristics employed to find local alignments within given sequences. Among the most successful heuristics to date are those used by FASTA and BLAST. Both programs use a heuristic that first preprocesses k-long exact matches in strings. Based off of these k-long alignments, the programs extend to search for local alignments. The genetic algorithmic approach does something similar but does not depend on k-long exact matches. Instead, it starts by randomly selecting possible local alignments. Based on the scoring techniques described in the Methods section, it evolves local alignment solutions until it produces solutions that are optimal or close to optimal.

Motif Finding:

The most successful motif finding heuristic is based on Gibbs sampling, a method essentially a special instance of genetic algorithms. Programs such as AlignACE randomly select a motif from a randomly selected sequence. They then compute some sort of alignment between the motif and randomly selected segments within given sequences (excluding the sequence the motif was chosen from). While genetic algorithms can be made to simulate the Gibbs Sampling technique, the genetic algorithm presented in this paper randomly generates a motif instead. This increases the chance of finding the "true motif" if it does not exist within the given sequences. The score returned by the objective function that evaluates each gene in the genetic algorithm can be made to return the log ratio of residue frequencies to background probabilities.

$$\sum_{i=1}^{N} \sum_{k=1}^{K} \log \frac{M(k, x^i_{a_i+k})}{Back(x^i_{a_i+k})} \qquad (1)$$

$$M_{kj} = \frac{1}{(N-1)+B} (\beta_j + \sum_{i=1}^{N} (x_{a_i+k} = j)) \qquad (2)$$

$$B = \sum_j \beta_j \qquad (3)$$

,where $x^1, x^2, ..., x^N$ are the given sequences and $a_1, a_2, ...a_N$ are comparison segments within $x^1, x^2, ..., x^N$ respectively. β_js are added pseudocounts to avoid taking the log of zero. This will match the algorithm employed by most Gibbs Sampling programs.

Another method is to do a global alignment between the randomly generated motif and the segments of the sequences. This gives the advantage that motifs can allow gaps or insertions in sequence comparisons. Sample pseudocode is given below:

For each pair of motif and subsequence a_i,
  do global alignment on motif and a_i
Sum up total scores

One of the fastest methods to compare motifs is to do a simple consensus-count that counts how many matches a motif gets per position in the given segments. This approach has the advantage of speed, and it still produces good results.

METHODS

This section will formulate solutions to the given problems discussed in the previous section.

Global Alignment

The description for the genetic algorithm implementation is as follows:

TABLEAU FOR GLOBAL ALIGNMENT

1.	**Objective**	Global Alignment of 2 sequences
2.	**Rep scheme**	
	Structure	pairs of bits
	Alphabet, Length	K = 2 (binary), L = $2(\|X\| + \|Y\|)$
	Mapping	Mapping: 01 $\rightarrow$, 10 $\downarrow$, 00/11 $\searrow$
3.	**Fitness**	Dynamic Programming Score
4.	**Param**	M=100, pCross=0.6, pMut=0.01
5.	**Termination**	End of Last Generation
6.	**Result Desig**	Final Best-of-generation

Table 1 - Global Alignment

The mapping from a 1-D binary gene to an optimal global alignment path is based on the dynamic programming array presented in the "Background" section. The mapping looks at binary digits (in a given gene) in pairs, each pair coding for one of three possible paths from the current cell to the next cell in the dynamic programming array. The three possibilities are coded as follows:

Binary Pair
- 01 right $\rightarrow$
- 10 down $\downarrow$
- 00/11 down and right $\searrow$

There are different ways to code these paths, but the one presented above is used because it is easy to understand and quick to implement. Since two bits are used to represent three possible paths, an extra state is available for additional information. For instance, this extra state could be used to permit non-penalizable gaps or deletions.
Gap penalty scoring is based on an arbitrary, user-defined gap function. Evaluating the score of each gene in a genetic algorithm run is fast because it is easy to calculate the score given a path.

Why it works:
i) Solution exists in gene space

The maximum length of a path is $|x| + |y|$, so constructing a gene containing $|x| + |y|$ pairs is sufficient to describe any solution. Any paths that hit either the right edge or bottom edge before going to the bottom-right cell will be truncated at those edges, and a path from the endpoint to the last cell will be concatenated. The scoring mechanism enforces this policy. In this way we can restrict the genetic algorithm to only produce valid global alignment paths.
ii) Evolving to an optimal path

The alignment score is a good indicator of optimal sequence alignments. It is likely that the optimal path has close-to-optimal subpaths. Thus, if a good path is found, it is likely that other good paths can be derived from the given good path. Mutation and crossover operators will cause the genetic algorithm to sample different local optimal groups. They have a similar effect to simulated annealing.

Local Alignment

The description for the genetic algorithm implementation is as follows:

TABLEAU FOR LOCAL ALIGNMENT

1.	**Objective**	Local Alignment of 2 sequences
2.	**Rep scheme**	
	Structure	binary string
	Alphabet, Length	K = 2 (binary), L = 4(sizeof(int))
	Mapping	Mapping: $X_{start}, X_{stop}, Y_{start}, Y_{stop}$
3.	**Fitness**	Global Alignment of subsequences
4.	**Param**	M=100, pCross=0.6, pMut=0.01
5.	**Termination**	End of Last Generation
6.	**Result Desig**	Final Best-of-generation

Table 2 - Local Alignment

In the model above, there is an assumption that the length of the sequences being compared can fit in an integer type. If that is not the case, modify the length of the gene to be the appropriate bit size. The binary string maps to four positions in which it will cut sequence X and Y. The resulting subsequences are then aligned using a global alignment algorithm, and the alignment score is returned.

Some Details: To ensure validity of the randomly generated positions, be sure to mod by $(|sequence| - 1)$ so that the position falls within the range of allowable values. Next, swap X_{start} and X_{stop} if X_{start} is greater than X_{stop}. Do the same for Y_{start} and Y_{stop}.
This method tends to produce large local alignments. If the goal is to produce smaller alignments, then instead of mapping the binary string to start and stop positions, map the "stop positions" to extendable lengths. Extendable lengths can code for how much longer than a defined length the local alignment can be. These extendable lengths need only be a few bits, depending on how much we want to be able to extend alignments.

Motif Finding

The description for the genetic algorithm implementation is as follows:

TABLEAU FOR MOTIF FINDING

1.	**Objective**	Find Motifs in sequences
2.	**Rep scheme**	
	Structure	binary string
	Alphabet, Length	K = 2 (binary), L = 114(see below)
	Mapping	(see discussion below)
		Extension size
		Generated Motif
		Motif Extension
		Start Positions
3.	**Fitness**	
	case 1	Dynamic Programming
	case 2	Consensus-counting
4.	**Param**	M=100, pCross=0.6, pMut=0.01
5.	**Termination**	End of Last Generation
6.	**Result Desig**	Final Best-of-generation

Table 3 - Motif Finding

Unlike many other motif finders, the above implementation does not select possible motifs from the given sequences. Instead, it generates the motif based on "good motifs" from previous generations. In the first run it randomly generates a motif. The advantage of this is that the "true motif" does not need to exist within the given sequences.

The length of a gene in the genetic algorithm depends on the number of sequences we are searching through and the allowable extension length for the motif. The length of a motif is defined as a constant. The extension length is how much we are willing to extend that length when generating the "true motif" that we will be scoring.
The size can be calculated by the following formula:
size = extension-size + motif-size $\times$ 2 + (2 $\times$ $(2^{extension-size})$) + number-of-sequences $\times$ size-of-int

The mapping structure is given here:

[extension size | motif | motif extension | start positions of subsequences]

The binary representation of the motif is taken in pairs since two bits are required to code for the four-letter alphabet (A,T,C,G).

The objective scoring function can be any comparison function for two sequences. Here we show the dynamic programming and "consensus-counting" approach described in the "Background" section.

Phase Shifts: In Gibbs sampling, since the comparison motif is chosen from the given sequences, a phase shift in the selection will produce a shift in the resulting motifs. To account for this, most Gibbs sampler techniques do a number of shifted comparisons when comparing the motif with selected subsequences. While the genetic algorithm implementation does not have the propagating shift problem, phase shifting when using the dynamic programming approach may help for boundary cases, depending on what type of gap score is used. The results in the next section include a phase shift of 2 positions in each direction.

Sample code for these implementations can be found at http://www.stanford.edu/~chongs/GA/

RESULTS

The following results are based on genetic algorithms described in the previous section.

Global Alignment

Trial #1:
Open Gap Penalty - 10
Extend Gap Penalty - 5
Match - 10
Mismatch - 3
Sequence1: ACCCCTTGGGAAATTGCCCATATTATA
Sequence2: CTTGCTTAAAAAATATCCGTT
Alignment:
ACCCCTTGGGAAATTGCCCATATTATA
CTTGCTTAAAAAATATCC$\diamond\diamond\diamond$GTT$\diamond\diamond\diamond$
Score: 40

Trial #2:
Gap Score - Sinusoid: 3 + sin(gap index)
Match - 10
Mismatch - 3
Sequence1: ACCCCTTGGGAAATTGCCCATATTATA
Sequence2: CTTGCTTAAAAAATATCCGTT
Alignment:
ACC$\diamond$CCTTGGGAAATTGCCCATATTATA
$\diamond$CTTGCTTA$\diamond$AAAAATATCC$\diamond$G$\diamond\diamond$T$\diamond$T$\diamond$
Score: 71.6

Trials #3 – 21: Same as trials as above except with different sequences and different gap scores. More trials can be found at http://www.stanford.edu/~chongs/GA/

All sequences with length $\leq$ 50 seemed to converge to an optimal path or close-to-optimal path within 100 generations (and 100 genes per generation). This is better than

the expected cubic run-time if we were to use dynamic programming for arbitrary gap scores.

Local Alignment

Sample Trial:
Gap Penalty - 5
Match - 10
Mismatch - 3
Sequence1: ACTTGCACTTTGCCCATATTATAAG-GCTTAGCCGTTTTCCCGAGATTAATAA
Sequence2: CTTGTTTCATATACTCG-GTTCAAATTGTATATCCGTT
Segment in X: from position 0 to 48
Segment in Y: from position 0 to 36

More trials on the website mentioned before.

Motif Finding

Sequence 1: ACTTGCACTTTGCCCATAT-TATAATATATATATATTAGCCGTTTTCCCGA-GATTAATAAATTACCGGATTACG
Sequence 2: CTTGTTTCATATACTCG-GTTCAAATTGTATATCCGTTATATATATACCTG-GCTTACTTTACCCGATTATAAAC
Sequence 3: ATGGCCTTATACCC-TATATATATAAATTATCTGCGCGTATACGTTATG-CATGGGGGACTCGGCAAATTACGGACGT
Sequence 4: CAACCAGCATTACTTACTGGTACG-GTACCCATTATATATATAGCAATATCGACTCG-GACTTAATATACCTAGATCTAGAGTAA

Motifs found by genetic algorithm under consensus-counting:
Motif #1: CCTTTCCCCTATT
Motif #2: TTTTTATATATA
Motif #3: ATATATTACTGGT
Motif #4: AGTTACCTATATA
Motif #5: TATACCTATAGTT

Motifs found by genetic algorithm under dynamic programming:
Motif #1: GATATAACTACGG
Motif #2: ATGTGTATATAAC
Motif #3: ATATATACCGTAG
Motif #4: CTTATGTATGATA

Using AlignACE 3.0
Motif 1
TATATATATA 0 24 1
TATATATATA 1 36 1
TATATATATA 2 14 1
TATATATATA 3 32 1
MAP Score: 12.6103

Motif 2
TATAATATGGG 0 12 0
TATATAACGGA 1 31 0
TTTATAATCGG 1 61 0
TATATATAGGG 2 11 0
TATATAATGGG 3 27 0
MAP Score: 5.84689

Motif 3
TGGGCAAAGTGCA 0 3 0
TATATATATTATA 0 19 0
TGAACCGAGTATA 1 9 0
TATATCCGTTATA 1 27 1
TAAGCCAGGTATA 1 42 0
TATATAGGGTATA 2 7 0
TGCATAACGTATA 2 37 0
TGGGTACCGTACC 3 18 0
MAP Score: 4.74264

Motif 4
ATATATTATA 0 19 0
GTATATGAAA 1 4 0
ATATATAACG 1 33 0
GTAAAGTAAG 1 51 0
ATATATATAA 2 15 1
ATACGTTATG 2 38 1
GTAAGTAATG 3 7 0
ATATATAATG 3 29 0
GTATATTAAG 3 58 0
MAP Score: 2.56997

Motif Finding on yeast genes(Saccharomyces cerevisiae):
Sequence #1: gggcgtgtggtctagtggtat-gattctcgctttgggcgacttcctgattaaacaggaaga-caaagcatgcgagaggccctgggttcaattcccagctcgcccc
Sequence #2: gggcacatggcgcagttggtagcgcgcttc-ccttgcaaggaagaggtcatcggttcgattccggttgcgtcca
Sequence #3: ggttgtttggccgagcggtc-taaggcgcctgattcaagaaatatcttgaccgcagt-gaactgtgggaatactcaggtatcgtaagatgcaagagttcgaatctcttagcaacca
Sequence #4: ggcaacttggccgagtggttaaggcgaaagatta-gaaatcttttgggctttgcccgcgcaggttcgagtcctgcagttgtcg
For the yeast, find motif in these first 4 anticodon regions:
Count Scoring
TTGCCCTCGCAG
GGTCGAAAGTCTC
Dynamic Programming
TTTCCGCATGTG
AGGTCCGAGCCTC

DISCUSSION OF RESULTS

Comparing the genetic algorithm approach for local alignments against leading heuristics such as those used by BLAST and FASTA, the genetic algorithm performs relatively well. The disadvantage of genetic algorithms is that it takes longer to find potential good local alignmnets. However, this is also a benefit since no preprocessing is needed to search for exact matches. As the number of consecutive matches grows larger, the preprocessing required for BLAST and FASTA grows exponentially. The genetic algorithm approach does not need this preprocessing step. The advantage of genetic algorithms is the method's generality. It allows arbitrary gap scores and does not require exact matches to be present. Algorithms such as those used by BLAST do not handle gaps very well. Using the genetic algorithm described in the "Methods" section, we can calculate arbitrary gap scores much easier.

The run-time for global alignment under genetic algorithms is:
O(GN), where G = number of generations and N = $\max(|X|, |Y|)$

This is linear if G is taken as a constant. The space requirement is $O(|X| + |Y|) = O(N)$. Thus, it is linear with respect to the longer sequence.

The space and time requirement for local alignment is the same. Thus, even if G grows linearly, the total run-time of $O(N^2)$ would still be significantly faster than the $O(N^3)$ dynamic programming run-time. The space requirement would still be linear with respect to N.

In many trials, the genetic algorithm performed as well as other heuristics employed by leading gene analysis tools today. In global and local alignments, genetic algorithms have the advantage that arbitrary gap scoring methods are easy to model. It is also easy to add position or sequence dependencies if necessary. In genetic algorithms, we need only change the objective function to take in position probabilities to incorporate positional dependencies in scoring alignments. The flexibility of genetic algorithms is the key difference between genetic algorithms and many other heuristics.

In motif finding, genetic algorithms essentially add to the Gibbs Sampling methods used today. With the correct objective function, the genetic algorithm approach will act just like a Gibbs sampler to find motifs. By increasing the generality of the algorithm (moving to genetic algorithms), more flexibility is given to introduce dependencies without having to change the underlying structure or state machine of the Gibbs algorithm. Crossover and mutation operations can be written to consider previous letters or letter frequency.

Out of the previously mentioned scoring methods in motif finding, consensus-counting is the quickest and produces "good" results. The motifs produced by consensus-counting are "good" in the sense that they are minimally distant from similar subsequences, where distant is defined by the number of mismatching letters per position.

The dynamic programming method of comparison in motif finding increases the time to search for motifs, but the added time grows by a constant factor since the maximum size of a motif is constant. While this increase in time may be noticeable for small sequences, for motifs with lengths far less than N, the dynamic programming method comes at a very low cost. The advantage of the dynamic programming method is that it can compare sequences while considering the possibility of insertions and/or deletions. If gap scores are increased toward infinity, the motifs produced by this dynamic programming method will approach the results produced by consensus-counting.

The time complexity of the dynamic programming based comparison is as follows:

$$\sum_{i=1}^{G}\sum_{j=1}^{P}\sum_{k=1}^{S}\sum_{l=-Phase}^{Phase} Align(motif, subsequence) \quad (4)$$

G = number of generations, P = size of population, S = number of sequences, Phase = number of phase shifts from initial target. Align(motif,subsequence) takes $O(M^2)$ time, where M = maximum size of motif. M and Phase are constant, but G or P may vary with N, where N = maximum length of a sequence.

CONCLUSION

This paper has demonstrated the applicability of genetic algorithms as feasible heuristics in computational genomics. More specifically, this paper has formulated solutions for sequence alignment and motif finding using genetic algorithms. These methods have highlighted genetic algorithm's flexibility to model otherwise complex models. The arbitrary gap scoring function, for instance, is not present in most other sequence alignment software.

FUTURE WORK

The success of genetic algorithms to align two sequences suggests that aligning multiple sequences will also be successful. While the number of cases that need to be

maximized for each step will increase exponentially with respect to the number of sequences, the number of bits required to encode the states, test cases, and transitions grows linearly.

There is a lot of potential in using genetic algorithm's flexibility to simulate positional dependencies in analyzing gene sequences. Currently, most programs have trouble introducing these dependencies without completely changing the underlying algorithm or state machine their algorithm depends on. Through genetic algorithms, many dependencis can be accounted for in the objective scoring function. Also, additional experimentation with genetic operators (like crossover and mutation) can potentially help genetic algorithms perform better.

Currently, many programs base their models for gene prediction on variants of Hidden Markov Models (HMMs). Instead of constructing these HMMs and training them with given data sets, it may also be feasible to construct biological models using genetic programming. Based on the constraints present in the data set, genetic programming can be used to evolve accurate gene prediction models. This approach may approximate certain biological phenomenon more accurately than what we currently believe "ought" to describe biology. However, we must still be careful because this approach is susceptible to the same over-training dangers HMMs are susceptible to. Since the entire construction is based on this data, it may be even more damaging to a genetic programming method.

REFERENCES

1. R. Durbin, S. Eddy, A. Krogh, G. Mitchison. 1998. *Biological Sequence Analysis.* Camridge, UK: Cambridge University Press

2. Koza, John R. 1992. *Genetic Programming - On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: The MIT Press

* Electronic address: chongs@cs.stanford.edu; URL: http://www.stanford.edu/~chongs/GA/