Multiple Sequence Alignment Using Evolutionary Programming

Kumar Chellapilla
Dept. of ECE
U. C. San Diego
La Jolla, CA
USA 92093-4007
kchellap@ece.ucsd.edu

Gary B. Fogel
Natural Selection, Inc.
3333 N. Torrey Pines Ct., Suite 200
La Jolla, CA
USA 92037
gfogel@natural-selection.com

Abstract- Multiple sequence alignment can be used as a tool for the identification of common structure in an ordered string of nucleotides (in DNA or RNA) or amino acids (in proteins). Current multiple sequence alignment algorithms work well for sequences with high similarity but do not scale well when either the length or number of the sequences is large or if the similarity is low. The focus of this paper is to develop an evolutionary programming (EP) algorithm for multiple sequence alignment. An EP method with representation specific variation operators is proposed and tested on several data sets. Comparisons to other algorithms suggests that this algorithm is well-suited to the multiple sequence alignment problem.

1. Introduction

Alignment of nucleotide or protein sequences is a fundamental process in molecular biology. Sequences that show a high degree of similarity across different taxa are thought to have similar structure and function. These types of sequences are important in deciphering the evolutionary history or phylogenetic relations among organisms (Sankoff et al., 1987). Sequence comparison can also be used to obtain information on secondary and tertiary structure (Hori and Osawa, 1979; Wong et al., 1976), and for estimating the evolutionary distance between the genomes of two organisms (Rempe, 1987).

The alignment of a pair of molecular sequences is a difficult problem in light of the fact that any two sequences in an ensemble may not be identical due to substitution, insertion or deletion of elements in either one or both of the two sequences at any position. These mutational processes are the root of change at the molecular level. One of the goals of any alignment procedure is to extract information from a collection of sequences despite their sometimes extensive evolutionary divergence.

Pairwise alignment attempts to compute the similarity between two sequences by minimizing the number of primitive mutational steps required to match the first to the second. Needleman and Wunsch (1970) were the first to solve the problem of maximum pairwise similarity alignment for two sequences. Rewards were given for similarities or identities in the two sequences and penalties were given for deletions (referred to as gaps) or mismatches. Dynamic programming (Sankoff, 1972; Waterman et al., 1976) has been accepted as an effective tool for the optimal alignment of two sequences. Waterman et al. (1976) generalized the method of Needleman and Wunsch (1970) to satisfy all the metric conditions (Sellers, 1974). His method was also capable of working with deletions and insertions of any length. Gotoh (1982) reduced the complexity of the method presented in Waterman et al. (1976) from $O(m_1^2 m_2)$ to $O(m_1 m_2)$ where m_1 and m_2 were the lengths of the first and second sequences. Myers and Miller (1988) demonstrated that Gotoh's method could be implemented in $O(m_1)$ space. However, the complexity still remains $O(m_1 m_2)$.

Multiple sequence alignment refers to the search for maximal similarity in three or more sequences (Chan et al., 1992). Direct extensions of pairwise alignment algorithms to the alignment of n sequences of length m have been offered. These method typically have $O(m^n)$ complexity (Chan et al., 1992) making them unsuitable when either the length or the number of sequences is large. Many approaches attempt to circumvent this increased computational complexity. ClustalW (Higgins and Sharp, 1988) uses the fast/approximate method of Wilbur and Lipman (1983) and the result of each comparison is a similarity score for a pair of sequences. The similarity scores are used to construct a dendrogram using the UPGMA cluster analysis method of Sneath and Sokal (1973). ClustalW is the most commonly used multiple sequence alignment tool available. In general, most algorithms require at least $n-1$ pairwise alignment processes when aligning n sequences (Zhang and Wong, 1997) and the majority use heuristics to arrive at high scoring non-optimal alignments that satisfy a number of constraints.

Godzik and Skolnick (1994) applied Monte Carlo methods to the multiple alignment of protein structures and sequences. Kim et al. (1994) proposed simulated annealing as the basis for an efficient multiple sequence alignment algorithm. Both of these approaches can accept arbitrary scoring functions including non-local ones. For small numbers of sequences with high similarity, these algorithms are slower than dynamic programming. However, as the number of sequences increases, their efficiency increases to a rate comparable or better than dynamic programming in generating near-optimal solutions.

Methods of evolutionary computation have been offered for the multiple sequence alignment problem (Notredame et al. 1996; Notredame et al. 1997; Zhang and Wong 1997; Anabarasu et al. 1998; Gonzalez et al. 1998; Notredame et al. 1998). In this paper, an evolutionary programming (Fogel, 1995) algorithm is proposed for multiple sequence alignment. This algorithm has been tested on several DNA sequence sets. Preliminary results suggest that this algorithm is viable and the quality of solutions is comparable to that obtained with ClustalW.

2. Method

For simplicity the description of the method will be limited to DNA sequences composed from the DNA nucleotide alphabet {A,T,C,G}. The same method can be directly extended for the alignment of RNA and protein sequences. Consider n DNA sequences of lengths $l_1, l_2, ..., l_n$ to be aligned. These n sequences were generally of different lengths. Any candidate alignment of these n sequences was represented as a matrix with the following properties:

1. The matrix had n rows, with the i th row representing the i th sequence.
2. All entries in the matrix consisted of elements from the set {A,T,C,G,–}, where A, T, C, and G represented the four DNA nucleotides, namely, adenine, thiamine, cytosine, and guanine. The symbol "–" infers a gap in the alignment such that there has been an insertion or deletion at that position in the sequence relative to the other sequences in the set. The non-gap entries in the i th row consisted of the symbols from the i th sequence taken in order.
3. The maximum number of columns was limited to $w = \lceil 1.2 l_{max} \rceil$, where $l_{max} = \max\{l_1, l_2, ..., l_n\}$and $\lceil x \rceil$ represents the smallest integer greater than or equal to x. The choice of 1.2 as a scaling factor allowed the alignment to be 20 percent longer than the longest sequence in the set. This choice was based on the observation that solutions to common alignment problems rarely contained more than 20 percent gaps.

Thus, the search space for alignments consisted of all possible matrices representing possible alignments and satisfying the above three conditions.

These alignments represented as matrices were optimized using evolutionary programming. A population of candidate alignments was maintained. A suite of variation operators was used to produce variation among existing alignments. Selection was used to determine which solutions were to survive to the next generation and which solutions were to be culled from the population. One application of variation and selection constituted a single generation. This process of variation and selection was iterated until a prespecified termination criterion was satisfied. The process is detailed below.

2.1 Initialization

Two methods of initialization were used based on the length and expected inter-sequence similarity of the sequences in the alignment. A population of μ initial parent alignment matrices was produced by randomly initializing their constituent rows. In each alignment, the i th row contained the i th sequence of length l_i. In the first method, the positions of these symbols in the row was chosen as follows: A random permutation of the numbers 1, 2, ..., w (representing column positions in the row) was produced. The first l_i entries in this permutation were chosen to be the positions for the symbols. The positions in the row represented by the remaining entries were filled with gaps. This process generated a uniform sampling from all possible alignments in the search space. The second method was based on precomputing all pairwise alignments for the sequences to be aligned. For each parent to be initialized, a random permutation of the n sequences was generated[1]. Pairwise global alignments of all adjacent sequences in the permutation were computed using the $O(n)$ space, $O(mn)$ complexity algorithm described in Myers and Miller (1988). These pairwise alignments were merged in a sequential manner starting with the first sequence in the permutation and ending with the last sequence. Two examples illustrating the initialization methods are shown in Figure 1. Associated with each alignment, A_i, were two self-adaptive parameters, σ_i, and η_i. σ_i represented the mean number of sequences to be selected for shuffle mutation and η_i represented the mean number of times each selected sequence was to be mutated. σ_i was initialized to $\lceil 0.2n \rceil$ and η_i was initialized to $\lceil 0.2w \rceil$. The generation number, g, was set to 1.

2.2 Variation

The μ parents were copied into μ offspring, with each parent generating a single offspring. Each of these offspring was probabilistically varied using one of five possible variation operators, namely, *RandomShuffle*, *LocalShuffleOne*, *GrowMatchedCol*, *RecombineMatchedCol*, *LocallyAlignBlock*. The corresponding probabilities of selecting each operator were 0.4, 0.3, 0.1, 0.1, and 0.1, respectively. While, the *RandomShuffle* operator was used for exploration, the *LocalShuffleOne*, *GrowMatchedCol*, *LocallyAlignBlock*, and *RecombineMatchedCol* operators were purely exploitative

1. Note that given n sequences, there are $n!$ number of possible permutations, each of which gives rise to a valid alignment that can be generated through the second method.

ID	Sequence	L	Permutation(1->10)	Positions	Sorted Positions	Initial Alignment
S1	ATCAA	(5)	3 5 2 6 9 1 7 4 8	3 5 2 6 9	2 3 5 6 9	-AT-CA--A
S2	TAATCAA	(7)	9 6 7 1 4 8 5 3 2	9 6 7 1 4 8 5	1 4 5 6 7 8 9	T--AATCAA
S3	ATCA	(4)	6 2 5 1 4 8 3 7 9	6 2 5 1	1 2 5 6	AT--CA---
S4	TAATCAT	(7)	7 4 9 1 3 5 8 6 2	7 4 9 1 3 5 8	1 3 4 5 7 8 9	T-AAT-CAT
S5	ATGATT	(6)	5 6 8 4 3 1 9 7 2	5 6 8 4 3 1	1 3 4 5 6 8	A-TGAT-T-

Figure 1(a). Example illustrating the random initialization procedure. S1 through S5 are five sequences of lengths (L) five, seven, four, seven, and six nucleotides, respectively. Initial alignment of these sequences was determined using w = ceil(1.2*7) = 9.

```
Permutation of the five sequences: 1 3 2 5 4
Pairwise alignments of adjacent sequences in permutation:
S1 ATCAA  S3 --ATCA-  S2 TAATCAA-  S5 --ATGATT
S3 ATCA-  S2 TAATCAA  S5 --ATGATT  S4 TAATCAT-
Merging
 Start     Add S2       Add S5      Final Alignment (Add S4)
S1 ATCAA  S1 --ATCAA  S1 --ATCAA-  S1 --ATCAA-
S3 ATCA-  S3 --ATCA-  S3 --ATCA--  S3 --ATCA--
          S2 TAATCAA  S2 TAATCAA-  S2 TAATCAA-
                      S5 --ATGATT  S5 --ATGATT
                                   S4 TAATCAT-
```

Figure 1(b). Example illustrating the pairwise global align based initialization method. S1 through S5 are five sequences of lengths five, seven, four, seven, and six nucleotides, respectively. For the simple example above the optimal multiple sequence alignment is directly obtained on initialization. However, with increasing number and length of sequences the initial alignment is not optimal but possesses high fitness.

(i.e., they guaranteed an offspring that was as good as or better than the parent).

The *RandomShuffle* operator picked p sequences at random from the alignment, A_i, for variation (see Figure 2). p was produced by sampling a Poisson random variable with mean σ_i. If p exceeded n or fell below one, it was reset to the limit that it violated. The selection of sequences was biased by the following scaling factor:

$$u_i = \frac{1}{2} + \frac{2}{5}\left(\frac{r_i - 1}{n - 1}\right) \tag{1}$$

where, r_i is the rank of the i th sequence in terms of its similarity to the consensus of the alignment. The consensus sequence was determined based on the number of occurrences of each nucleotide, A, T, C, and G in each of the columns of the alignment (standard IUPAC redundant nucleotide letter codes).

Each of the p selected sequences was varied as follows:

1. The number of symbols to be shuffled, q, was obtained by sampling a Poisson random variable with mean η_i. If q exceeded w or fell below one, it was reset to the limit that it violated.
2. q symbols were randomly shuffled with gaps. Randomly shuffling a symbol comprised selecting a gap at random, extracting the gap from the sequence, and inserting the gap to the left or right of the symbol. If the selected gap was to the right of the symbol, then it was inserted to the left the symbol and vice versa. The selection of symbols and gaps was biased by the scaling factors v_{sj} and v_{dj}, respectively, defined by

$$v_{sj} = \alpha_j - (\alpha_j - \gamma_j)\left(\frac{M_j - 1}{n - \beta_j - 1}\right) \tag{2}$$

$$v_{dj} = 1 - v_{sj} \tag{3}$$

where

$$\beta_j = min(N_j - M_j, 3) \tag{4}$$

$$\alpha_j = 1 - 0.25(1 + \beta_j) \tag{5}$$

$$\gamma_j = 1 - \alpha_j \tag{6}$$

$$N_j = \text{number of symbols in the } j \text{ th column} \tag{7}$$

$$M_j = \text{number of matches in the } j \text{ th column} \tag{8}$$

Lognormal self-adaptation was used to update σ_j and η_j parameters of the offspring before p and q were sampled for variation. Mathematically,

$$\sigma_j' = \sigma_j exp\left(\frac{R_j(0, 1)}{\sqrt{2}}\right) \tag{9}$$

$$\eta_j' = \eta_j exp\left(\frac{R_j(0, 1)}{\sqrt{2}}\right) \tag{10}$$

where R_j represents a Gaussian random variable with zero mean and unit variance that is resampled for every σ_j' and η_j.

The *LocalShuffleOne* operator selected one of the sequences at random and scanned through it to find all symbols that had one or more gaps adjacent to them (see Figure 2). One of these symbols was selected with uniform probability. Notice that the selected symbol can be shifted to the left or right (or both) due to the presence of adjacent gaps. The selected symbol was shifted to each of its neighboring gaps and the fitness of the alignment was recomputed. The

Operator	Parent(s)	Offspring	Description
RandomShuffle	123456789 -AT-CA--A T--**AATCA**A AT--CA--- T-AAT-CAT A-**TGAT**-T-	123456789 -AT-CA--A T-**AATCA-**A AT--CA--- T-AAT-CAT A--**TGAT**T-	The second and fifth sequences have been randomly shuffled. The symbol (gap) at the beginning (ending) of the fifth (second) sequence were selected at random (in a biased manner) from the available symbols (gaps) in the selected sequence. The subsequence indicated in bold has been modified due to the shuffling of the selected symbol and gap.
LocalShuffleOne	123456789 -AT-CA-AA T--AATCAA AT--C**A**--- T-AAT-CAT A-TGAT-T-	123456789 -AT-CA-AA T--AATCAA AT--C--**A**- T-AAT-CAT A-TGAT-T-	The third sequence was selected for variation uniformly at random from the five sequences. The fourth symbol (A) was selected at random. The symbol A can be shifted to three possible neighboring positions in columns seven, eight, and nine respectively. Of these three, shifting to the eight column was chosen as it resulted in the highest increase in fitness.
GrowMatchedCol	123456789 -AT-C**A**-**T**C TA-C-**AT**AA AT--C**A**--**T** T-C--**A**--**T** A-TC-**A**-**T**-	123456789 -AT-C**AT**-C TA-C-**AT**AA AT--C**AT**-- T-C--**AT**-- A-TC-**AT**--	In the parent, the sixth column is fully matched with A's. The *GrowMatchedCol* operator selects one such matched column and tries to generate matched columns adjacent to it. For the sixth column, there are two possible fully matched adjacent columns that can be generated, one consisting of all C's to its left and the other consisting of all T's to its right. The column of T's to the right is randomly selected (with equal probability) for generation.
Recombine-MatchedCol	Parent1 123456789 ----●---- -ATC**A**--AT T--A**A**TCAA ----**A**TCA- T-AA**A**-C-T A-TG**A**T-T- Parent2 1234567890 ------*--- -ATCAA**T**--- T--A-A**T**CAA -----A**T**CA- T-AAAC**T**--- A-TG-A**T**-T-	Offspring 123456789012 ----*---*--- -ATC**A**--A**T**--- T--A**A**---**T**CAA ----**A**---**T**CA- T-AA**A**-C-**T**--- A-TG**A**---**T**-T-	Column five in the first parent consists of all A's and is fully matched. Similarly, column seven is fully matched in the second parent. *RecombineMatchedCol* tries to generate an offspring that contains both the matched columns. In the offspring, the arrangement of the symbols in each sequence is the same as that of the first parent up to the T's in the matched column of the second parent. The arrangement of the sequences after the matched T's in the offspring is the same as that in the second parent. Extra gaps are inserted in a copy of the first parent to line up the T's as they appear in the second parent. In the general case, when a number of possible columns can be recombined, one is selected with uniform probability for generation in the offspring. Matched up columns in the second parent that require breaking up one or more matched columns in the first parent are discarded.

Figure 2. Examples illustrating the first four variation operators, *RandomShuffle*, *LocalShuffleOne*, *GrowMatchedCol*, and *RecombineMatchedCol*. S1 through S5 are five sequences of lengths five, seven, four, seven, and six nucleotides, respectively.

position of the symbol that gave the highest fitness became the destination for the symbol. If none of the neighboring gap positions generated a better fitness, then the sequence was left unchanged.

The *GrowMatchedCol* operator selected a fully matched column in the alignment (with no gaps) with no more than one adjacent matched column and attempted to add to the matched column by generating, if possible, another matched column next to it (see Figure 2).

The variation of an alignment using *RecombineMatched-Col* comprised the following steps:

1. Select a mate at random with uniform probability from the population.
2. Identify all matched columns in the current alignment and the random mate.
3. Determine all matched columns in the mate that are not present in the current alignment and can be added to it without disrupting any existing matched columns. One of these matched columns in the mate was regenerated in the current alignment by lining up the corresponding symbols in the current alignment in one column.

The *RecombineMatchedCol* operator (see Figure 2) was fail-safe in that the number of matched columns was guaranteed not to decrease. However, the process of lining up the symbols to generate a new matched column could result in a reduction in the overall fitness of the alignment.

The *LocallyAlignBlock* operator was used to speed up the evolution of sub-alignments between fully matched columns. It identified all sub-alignments that were located between matched columns and locally aligned one of these sub-alignments selected at random. The local alignment procedure was a modified version of the alignment construction algorithm presented in Zhang and Wong (1997):

```
1. Extract the sub-alignment and replace it with gaps. This
   extracted and filled in region of the alignment was
   referred to as the gap sub-alignment. Let the width of
   the gap sub-alignment be $w_g$.
2. Convert the extracted sub-alignment into a set of subsequences
   by removing all gaps.
3. Find the longest subsequence, say row $i$, and insert it
   into the alignment (without any gaps).
4. For $j = 1$ through $n$ and $j \neq i$,
   a. Place the $j$ th subsequence at the beginning of the
      $j$ th row of the gap sub-alignment. Let the length of
      the subsequence be $s$.
   b. for $k = s$ down to 1,
         Move the $k$ th symbol in the subsequence in the
         free space (to its right) of the gap sub-alignment,
         to the position that maximizes fitness.
      endfor
   endfor
```

Notice that the above algorithm will always generate the same arrangement of the sub-alignment. Further, the optimized sub-alignment is not guaranteed to be global as it depends on the order in which the subsequences are placed and optimized in the gap sub-alignment (step 4). Further, if a sub-alignment that was previously optimized using *LocallyAlignBlock* was selected, then a second application of the operator would not generate any variation. In view of this, a randomized version was used that picked the j values in step 4 in a randomly permuted order.

2.3 Fitness Evaluation

Before computing the alignment fitness, all columns that consisted of only gaps were deleted. The fitness of the resulting cleaned up alignment was computed using

$$\text{Fitness} = \mathit{SymbolScore} - \mathit{GapScore} \qquad (12)$$

where *SymbolScore* was the overall score for the number of matched symbols over all columns and the *GapScore* was the overall score for the number of gaps over all columns. The *SymbolScore* and *GapScore* values were computed using

$$\mathit{SymbolScore} = \sum_j M_j\left(1 + \frac{M_j}{n}\right) \qquad (13)$$

$$\mathit{GapScore} = \sum_j (n - N_j)\left(1 + \frac{n - N_j}{n}\right) \qquad (14)$$

The number of matches in each of the columns was linearly scaled (such that any column that was fully matched up was doubled) and the aggregate sum of all such scaled number of matches over all columns became the *SymbolScore*. Similarly, the *GapScore* was computed based on linearly scaled number of gaps over all columns. Fitness was to be maximized, i.e., alignments with a higher fitness were considered to be better.

2.4 Selection and Termination

Tournament selection as is typically implemented in evolutionary programming (Fogel, 1995) was chosen to determine which individuals in the current population were to become parents for the next generation. Each member of the population was compared with 10 opponents that were randomly selected (with replacement) from the population. For each comparison in which the fitness of the member was equal to or higher than that of the opponent, the member received a win. The μ members with the highest number of wins were selected to be the parent alignments for the next generation. The generation number, g, was incremented by one.

The process of variation, fitness evaluation, and selection was repeated until one of three termination criteria were satisfied:

1. The number of generations exceeded $g_{max} = 200$.
2. The best fitness did not improve over 100 generations.
3. The number of gaps in the best alignment fell below 0.2%.

3. Results

Several multiple sequence alignment experiments with varying number and lengths of sequences were conducted to test the proposed evolutionary programming method. In the interests of space, results regarding only four data sets are briefly presented here[2]. These data sets differed with respect to their length, number, or similarity.

Data set 1 was composed of 10 sequences from Zhang and Wong (1997). These 10 sequences were very similar and had been directly compared by Zhang and Wong to ClustalW in a previous analysis (Zhang and Wong, 1997). The average length of data set 1 was 212 nucleotides. Data set 2 was composed of 8 sequences of 16S rRNA (acquired from GenBank). These sequences were equally similar to the 10 sequences in data set 1, however the average length was significantly larger at 457 nucleotides. Data set 3 was composed of the 5' portion of the histone H3-II gene from 21 species of *Tetrahymena*, a freshwater ciliate (Brunk and Sadler, 1990). These sequences were of nearly the same length as those in data set 1, possessed a high similarity, but had twice the number of sequences relative to data set 1. Data set 4 contained 21 sequences from a 200 nucleotide intergenic region between histone genes H3-II and H4-II in *Tetrahymena*. This region was previously characterized as

2. The full set of results are available at http://vision.ucsd.edu/~kchellap/

having a much lower similarity in comparison to the sequences in data set 3 (Brunk and Sadler, 1990). Data set 4 therefore has the lowest similarity, with the same number of sequences as in data set 3, but with sequence lengths that were approximately three times longer.

Table 1 summarizes the results obtained using the proposed EP algorithm for sequence alignment. On the first data set, S1, the EP approach discovered the same solution as that found by Zhang and Wong (1997) and ClustalW. The number of matched columns was 198 (Zhang and Wong (1997) incorrectly stated that ClustalW's solution had only 197 fully matched columns). The first data set demonstrates that this technique is equally robust as ClustalW and the method presented by Zhang and Wong (1997). Unfortunately, further comparisons with the method of Zhang and Wong (1997) were not possible due to the non-availability of the remaining data sets reported in their paper. Therefore, we were forced to make direct comparisons only to ClustalW.

Data set 2 contained 8 sequences of 16S rRNA from a variety of bacteria. These sequences were roughly twice the mean length of the sequences in data set 1 and contained roughly the same number of sequences. In both cases, the best alignment (449 matched columns) was discovered by both the EP algorithm and ClustalW. Data set 3 contained sequences from the histone H3-II gene from 21 species of *Tetrahymena.* These sequences were roughly half the length of data set 1. Again, the best alignment (109 matched columns) was discovered by both algorithms.

As a true test of the performance of the EP algorithm relative to ClustalW, 21 sequences with a mean length of 333.4 nucleotides were used in data set 4. These sequences compose the intergenic region between histone genes H3-II and H4-II in various species of the ciliated protozoa *Tetrahymena.* This region has been previously identified as having less sequence similarity than either the histone H3-II or histone H4-II genes (Brunk and Sadler, 1990; Brunk et al., 1990). When using both EP and ClustalW, EP was able to discover a better alignment (102 matched columns relative to 91 matched columns) in only 180 generations. This difference suggests that alignment algorithms using evolutionary computation are likely to outperform ClustalW when the similarity of the sequences is low.

4. Discussion

Three variables are particularly important for any multiple sequence alignment algorithm; the number of sequences, the average length of the sequences, and the overall similarity of the sequences. Traditional algorithms such as ClustalW are known to be very successful when the number or average length is low and the overall similarity of the sequences is high. For instance, ClustalW can determine the optimal alignment for the sequences in data set 1 (10 very similar sequences of average length 211 nucleotides) in 27 seconds. However, this same algorithm takes 14 hours and 3 minutes to compute the alignment for 10 less similar sequences of average length 9000 nucleotides (Zhang and Wong, 1997). Even after using ClustalW, most biologists use some degree of post-processing to refine the alignment into something more "meaningful." The reason for this lies in the limitations ClustalW poses on the type of objective functions that it can optimize. Clearly, an evolutionary approach that relaxes this limitation and allows for any arbitrary user defined fitness function could be used to search the space of possible sequence alignments in a more efficient manner.

Notredame et al. (1997) used a genetic algorithm (RAGA) for aligning related sequences of RNA using information regarding their secondary structure. Information about the secondary structure of one of the two sequences was used to predict the position of the structural elements in the second sequence. RAGA was an extension of an earlier algorithm (SAGA) for multiple sequence alignment using genetic algorithms (Notredame and Higgins, 1996). The algorithm for RNA structural alignment was also made parallel (PRAGA) (Notredame et al., 1997). Although these efforts are a powerful step in the right direction, RAGA and PRAGA make use of structural information to assist in their alignment. With a very limited number of RNA structures determined by X-ray crystallography or NMR, structural information is usually predicted through the use of energy minimization algorithms. Energy minimization algorithms have been determined to predict incorrect structures in some cases (Fields and Gutell, 1996) and therefore, alignment on the basis of structural details is only as valid as the pre-determination of the structure using thermodynamics. The method presented in this paper is useful when structural information is not available or is not desired.

Zhang and Wong (1997) developed a genetic algorithm approach to multiple sequence alignment. Scoring of the alignment was based on the number of fully matched columns. Their method was directly compared with ClustalW. Their approach focused on the identification of matched columns and mutation between columns that were found by a pre-alignment tool. The method works well when the optimal alignment that is being searched for contains a large number of fully matched columns i.e., when there is a high similarity between the sequences. However, few fully matched columns will be found in sequences of low similarity, making this algorithm useful only for very similar sequences. Initialization by pre-alignment for matched column discovery places the search close to local optima on the response surface. Evolutionary computation will easily discover the nearest local optima, but must be able to escape the local optima in order to discover the global optima. In the experiments described by Zhang and Wong (1997), the

Table 1: Data sets used for testing the proposed evolutionary programming procedure for multiple sequence alignment. Information regarding the data sets is provided in the Appendix.

Data Set	Number of Sequences	Mean Sequence Length in nucleotides (min,max)	Percentage of matched columns based on the best alignment using ClustalW	Number of Matched Columns in ClustalW solution	Number of Matched Columns in EP solution	Number of Generations	EP Score
1. S1 (Zhang, 1997)	10	211.9 (211, 212)	93.39	198	198	200	4082
2. 16S rRNA	8	457.0 (457, 457)	98.25	449	449	400	7233
3. Histone H3	21	122.0 (122, 122)	89.34	109	109	160	4766
4. Histone H3-H4 Intergenic Region	21	333.4 (322, 346)	27.41	91	102	180	7033

genetic algorithm approach was terminated after 10 generations of stagnation. It is likely that with decreasing sequence similarity, 10 generations may not be sufficient to escape multiple local optima required to discover the global optimal alignment. A special mutation operator was developed to make drastic changes to the alignment and force solutions out of local optima.

Gonzalez et al. (1998) generated a simulation of multiple protein sequence alignment using genetic algorithms. The three example data sets used in their analysis contained short sequences (30 nucleotides or less) and a small number of sequences (three sequences). The results were subjectively compared to traditional algorithms, however it is expected that ClustalW could have been easily used on such a small data set with equal performance in equal or less time. The true benefits of any evolutionary computation approach to the multiple sequence alignment problem will be to succeed where other algorithms fail, rather than to equal the performance in areas where traditional algorithms are known to succeed. For short sequences that are few in number, ClustalW already presents a method that is widely appreciated for solving the problem in a rapid fashion.

Anabarasu et al. (1998) generated a multiple sequence alignment algorithm using a parallel genetic algorithm. This algorithm was tested on 4 protein sequence data sets of lengths between length 48 and 292 nucleotides. The number of sequences ranged from 4 to 15. Although able to generate roughly similar or even lower scores than ClustalW, the operational time of their approach was longer than ClustalW by two orders of magnitude. The similarity of the sequences in the data sets was not reported and could not be determined from the data provided in the paper.

All previous attempts at multiple sequence alignment using evolutionary computation have focused on two central themes; a genetic algorithm approach with emphasis on crossover including, to a lesser degree, special mutation operators, and comparison of this approach to ClustalW on small sequence sets with high similarity. However, any sequence alignment algorithm must trade off computation speed for alignment accuracy, especially when there is a low similarity between the sequences that are to be aligned. A poor alignment that is generated in a rapid fashion is still a poor alignment. If there is any part of the multiple sequence alignment problem where evolutionary computation can be useful, it is not in the alignment of small sets of very similar sequences but in large sets of sequences with low similarity. It is these types of alignments that have been missing from the literature to date. Generally, it is also difficult to directly compare the outputs of ClustalW and the various EC approaches because of their different scoring schemes. One comparative measure of success that can be used is the number of matches and the number of fully matched columns (i.e., columns with only A or T or G or C) in the alignment, as has been used in this study.

Future experiments will use additional data sets (Briffeuil et al., 1998) to compare the power and confidence against multiple sequence alignment servers on the internet. A revised nucleotide scoring matrix (or PAM matrix in the case of protein sequences) will be incorporated to make the alignment score more realistic rather than rely solely on the number of matches. At present, the EP approach treats all mismatches with equal penalty. However, the frequency of transitions or transversions may not be equal in the sequences that are being compared. It is hopeful that this altered scoring scheme will allow even better alignments to be discovered.

5. References

Anabarasu, L.A., Narayanasamy, P., and Sundararajan, V. (1998) Multiple sequence alignment using parallel genetic algorithms. In: The Second Asia-Pacific Conference on Simulated Evolution And Learning (SEAL-98), X. Yao, R. McKay, C.S. Newton, J.H. Kim and T. Furuhashi (eds.).

Brunk, C.F. and Sadler, L.A. (1990) Characterization of the promoter region of *Tetrahymena* genes, Nuc. Acids Res. 18(2):323-329.

Brunk, C.F., Kahn R.W., and Sadler, L.A. (1990) Phylogenetic relationships among *Tetrahymena* species determined using the polymerase chain reaction, J. Mol Evol. 30:290-297.

Briffeuil, P., Baudoux, G., Lambert, C., DeBolle, X., Vinala, C., Feytmans, E., and Depiereux, E. (1998) Comparative analysis of seven multiple protein sequence alignment servers: clues to enhance reliability of predictions, Bioinformatics 14(4):357-366.

Chan, S.C., Wong, A.K.C., and Chiu, D.K.Y. (1992) A survey of multiple sequence comparison methods, Bull. Math. Biol. 54(4):563-598.

Fields, D.S. and Gutell, R.R. (1996) An analysis of large rRNA sequences folded by a thermodynamic method. Current Biology 1:419-430.

Fogel, D.B. (1995) Evolutionary Computation: Towards a New Philosophy of Machine Intelligence, IEEE Press, New York.

Godzik, A. and Skolnick, J. (1994) Flexible algorithm for direct multiple alignment of protein structures and sequences. CABIOS 10(6):587-596.

Gonzalez, R.R., Izquierdo, C.M., and Seijas, J. (1998) Multiple protein sequence comparison by genetic algorithms. In: Proceedings of SPIE: Applications and Science of Computational Intelligence (SPIE-98), S.K. Rogers, D.B. Fogel, J.C. Bezdek, and B. Bosacchi (eds.). pp. 99-102. SPIE-The International Society for Optical Engineering, Bellingham, Washington.

Gotoh, O. (1982) An improved algorithm for matching biological sequences, J. Mol. Biol. 162:705-708.

Higgins, D.G. and Sharp, P.M. (1988) CLUSTAL: a package for performing multiple sequence alignments on a microcomputer, Gene 73, 237-244.

Hori, H. and Osawa, S. (1979) Evolutionary change in 5SRNA secondary structure and a phylogenetic tree of 54 5SRNA species, Proc. Natl. Acad. Sci. USA 76:381:385.

Kim, J., Pramanik, S. and Chung, M.J. (1994) Multiple sequence alignment using simulated annealing. CABIOS 10(4):419-426.

Myers, E.W. and Miller, W. (1988) Optimal alignments in linear space, CABIOS 4(1):11-17.

Needleman, S.B. and Wunsch, C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence for two proteins, J. Mol. Biol. 48:443-453.

Notredame, C. and Higgins, D.G. (1996) SAGA: sequence alignment by genetic algorithm. Nuc. Acids Res. 24:1515-1524.

Notredame, C., O'Brien, E.A., and Higgins, D.G. (1997) RAGA: RNA sequence alignment by genetic algorithm. Nuc. Acids Res. 25:4570-4580.

Notredame, C., Holm, L., and Higgins, D.G. (1998) COFFEE: an objective function for multiple sequence alignments. Bioinformatics 14(5):407-422.

Rempe, U. (1987) Characterizing DNA variability by stochastic matrices, *in* Classification and Related Methods of Data Analysis, H.H. Bock (Ed.) Amsterdam: Elsevier.

Sankoff, D., Abel, Y., Cedergren, R.J., and Gray, M.W. (1987) Supercomputing for molecular cladistics, *in* Classification and Related Methods of Data Analysis, H.H. Bock (Ed.). Amsterdam: Elsevier.

Sankoff, D. (1972) Matching sequence under deletion-insertion constraints, Proc. Natl. Acad. Sci. USA 64:4-6.

Sellers, P. (1974) An algorithm for the distance between two finite sequences. Comb. Theory 16:253-258.

Sneath, H.A. and Sokal, R.R. (1973). Numerical Taxonomy. San Francisco: W.H. Freeman.

Waterman, M.S., Smith, T.F., and Beyer, W.A. (1976) Some biological sequence metrices. Adv. Math 20:367-387.

Wilbur, W.J. and Lipman, D.J. (1983) Rapid similarity searches of nucleic acid and protein data banks. Proc. Natl. Acad. Sci. USA 80:726-730.

Wong, A.K.C., Liu, T.S., and Wang, C.C. (1976) Statistical analysis of residue variablility in cytochrome C, J. Mol. Biol. 102:287-295.

Zhang, C. and Wong, A.K.C. (1997) A genetic algorithm for multiple molecular sequence alignment, CABIOS 13(6): 565-581.

Appendix

Data Set 1: DNA, S1 (Zhang and Wong, 1997)

HCV2l1A10	HCV2L3A5	HCV2L3A7
HCV2L3A9	HCV2L3B1	HCV2L3B2
HCV2L3C1	HCV2L3C8	HCV2L3D4
HCV2L3E6		

Data Set 2: RNA, 16S rRNA (GenBank)

AF095268	AF095267	AF095266
AB023287	AB023286	AB023285
AB023284	AB023283	AB023279
AB023278	AB023276	

Data Set 3: DNA, Histone H3 (Brunk et al., 1990)

TPAHISIN	TNIHISIN	TNHISIN
TMIHISIN	TMHISIN	TLHISIN
THHISIN	TFHISIN	TEHISIN
TCUHISIN	TCHISIN	TCAHISIN
TBHISIN	TAUHISIN	TAHISIN
TTHISIN	TSHISIN	TRHISIN
TPYHISIN	TPIHISIN	TPHISIN

The first 122 symbols from each of the above sequences were used for alignment.

Data Set 4: DNA, Histone H3-II, Histone H4-II intergenic region (Brunk et al., 1990). The sequences were the same as those in Data Set 3. The intergenic sequence was extracted starting at 123rd symbol. The length of the extracted intergenic sequences were 329, 328, 332, 337, 344, 336, 334, 325, 335, 342, 334, 333, 335, 330, 333, 324, 334, 334, 334, 333, 348, respectively.