Genetic Algorithm Based Approach for Nucleic Acid Pattern Extraction

Miao Liu
University of Nebraska, miaoliu@mail.unomaha.edu

Hai-Feng Guo
University of Nebraska at Omaha, haifengguo@mail.unomaha.edu

Zhengxin Chen
University of Nebraska at Omaha, zchen@mail.unomaha.edu

Genetic Algorithm Based Approach for Nucleic Acid Pattern Extraction

Miao Liu
Dept. of Computer Science
University of Nebraska
Omaha, Nebraska 68182
miaoliu@mail.unomaha.edu

Hai-Feng Guo
Dept. of Computer Science
University of Nebraska
Omaha, Nebraska 68182
haifengguo@mail.unomaha.edu

Zhengxin Chen
Dept. of Computer Science
University of Nebraska
Omaha, Nebraska 68182
zchen@unomaha.edu

ABSTRACT

Finding a common pattern among nucleic acid sequences in a given database is an important yet relatively difficult problem in computational biology. Such a pattern is useful for describing the characteristics of a certain family of nucleic acid sequences, and can also be used for classification purposes as well as examine the closeness of two organisms. In this paper, we present a global pattern extraction tool named GAPE which can be applicable in computational biology to describe a certain family of nucleic acid sequences with common features. The algorithm utilizes an optimized *Genetic Algorithm* (GA) framework to drive the evolution of desirable patterns. A specialized pair-wise alignment algorithm is also introduced to efficiently examine the closeness of a sequence to a regular expression pattern. Experimental results using real biological data are shown to indicate the effectiveness of the tool.

Keywords

Patten extraction, motif discovery, classification, genetic algorithm

INTRODUCTION

Functionally related nucleic acid sequences share some common elements which correspond to residues conserved during evolution due to their important structural or functional roles (Rigoutsos and Floratos, 1998; Hertz and Stormo, 1999). This property allows researchers to identify matching nucleic acid sequences that have similar functional behaviors as a given sequence or pattern. This type of operation is referred to as *pattern matching* in the computational biology. Pattern matching problems has been well studied and various tools are available, include alignment algorithms (Needleman and Wunsch, 1970) and BLAST (Altschul, Gish, Miller, Myers and Lipman, 1990). Patterns are not only limited to concrete nucleic acid sequence strings. Rather, researchers allow patterns to contain regular expression operators to increase their expressive power and flexibility (Brazma, Jonassen, Eidhammer and Gilbert, 1998; Conklin and Farrah, 1998). Other extensions (Quest and Ali, 2004) also provide a mechanism for users to define the number of errors allowed, where different numbers can be used to control the degrees of similarity of imperfect matches allowed.

In addition to pattern matching, another useful operation based on the property of common elements shared in nucleic acid sequences is to find a global pattern of a family of sequences that are functionally related. Global pattern is essentially a representative of a given family. It contains the common local motifs discovered in the family, and the expressions used to connect these motifs due to the variations in between them. These variations may be expressed in forms of some grammar (e.g. regular expression). Global patterns would be desirable for example if we can find a pattern of DNA sequences of people who carry a certain type of cancer. This pattern can then be used to compare against other sequences to determine whether they carry the same type of cancer or not. We call the process of generating the common non-contiguous global pattern the *pattern extraction* problem.

Pattern extraction problems can be understood as a reverse process of pattern matching. Unfortunately, the algorithms used to solve pattern matching problems are irreversible, thus cannot be applied to solve pattern extraction directly. Most of the tools available for pattern discovery/extraction mainly concentrate at local levels, which are often referred to as *motif discovery* tools. They employ either a profile or pattern representation of a motif. In a profile-based method, a motif is represented by a position-specific scoring matrix. Packages that employ this type of representation include GibbsDNA (Lawrence, Altschul, Boguski, Liu, Neuwald and Wootton, 1993), MEME (Bailey and Elkan, 1995) and CONSENSUS (Hertz and Stormo, 1999). In a pattern-based method, a motif is represented as a string. These strings may contain mismatching information or even regular expression operators. Packages that employ pattern representation include TEIRESIAS (Rigoutsos and Floratos, 1998), WINNOWER and SP-STAR (Pevzner and Sze, 2000). Another form of pattern-based method recently introduced by Buhler and Tompa named Projection uses locality-sensitive hashing to search for motifs (Buhler and Tompa, 2002; Raphael, Liu and Varghese, 2004).

Since the above methods concentrated on local motifs, they may not be suited to extract the global pattern of a nucleic acid family. In this paper, we present an optimized *Genetic Algorithm* (GA) based approach to extract non-contiguous global pattern of a given nucleic acid family. The GA simulates an evolutionary process to optimize the strings containing the pattern information of this family during the process. The optimizations of the GA can be described as follows: the initial patterns are generated through an efficient *cluster analysis* procedure in order to increase accuracy, where each individual pattern is a continuous string representing some common elements found in the family; genetic operators named *mutation*, *crossover* and *fusion* take sufficient advantages of alignment results among nucleic acid sequences to drive the evolution of desirable patterns, and they ensure the common local motifs obtained in the current patterns are carried out to the next generation; in addition, a post-GA procedure is applied to generate a transformed pattern in a regular expression by incorporating the GA-generated pattern with the original nucleic acid sequences in the family. The patterns obtained by our method contain local motifs along with regular expression operators, and it may be further processed to extract local motifs as well. In addition, the obtained global pattern can be used to search for matching sequences within nucleic acid databases, or to test the membership of new sequences for classification purposes.

One of the tools available for protein and nucleic acid sequences classification is called HMMER (Eddy, 1998). It is a biological sequence analysis tool which uses profile Hidden Markov Models (Eddy, 1998) as its underlying structure. The idea of HMMER is to study the sequences from a given database in order to construct a hidden Markov model (HMM) profile for classification. New sequences to be classified are sent to the profile and scores are computed based on the probabilities. Different from HMMER, our tool takes a pattern based approach for classification. Since the pattern obtained by our extraction algorithm optimally describes the characteristics of a given database, new sequences can be compared against this pattern to examine whether they carry the same characteristics as the pattern. Scores are assigned to each comparison, which are used to estimate the membership of the tested sequences.

Most of the components used in our tool are implemented using Java, with the exception of a *multiple sequences alignment* (MSA) tool named ClustalW (Thompson, Gibson and Higgins, 1994). We have chosen to use ClustalW because it is well developed and widely accepted by many researchers. We examined the effectiveness of the tool using a real biological database Rfam obtained from (Jones, Bateman, Marshall, Khanna and Eddy, 2003). Pattern extraction and classification are applied on eight distinct families in the database. The average classification accuracy among these eight families is 99.5%.

GENETIC ALGORITHM BASED PATTERN EXTRACTION (GAPE)

Given a set of nucleic acid sequences, the problem of pattern extraction is to find an optimal *global* pattern such that it can be used to express the common features of these sequences. The extracted pattern can serve for classification purposes to determine whether a given sequence is within this family or not, which is very useful in computational biology. The challenge, however, is that optimal patterns may not be easily discovered due to huge data sizes and lack of knowledge. Therefore, finding optimal approximations of such patterns is the objective.

A tool, called *GAPE*, is introduced for pattern extraction in this paper. The overview of GAPE can be illustrated using Figure 1. Each oval in the figure represents a component of the tool. *Data Management* gives users ability to organize nucleic acid data. For example, nucleic acid data needs to be represented in FASTA format and then stored into *data repository*. Once the data file is successfully loaded into the system, *pattern extraction* may be performed. The final output is organized and displayed back onto the interface and the next set of operations may continue. GAPE also gives users abilities to perform nucleic sequences classification and further extract local motifs as well, based on the obtained pattern.

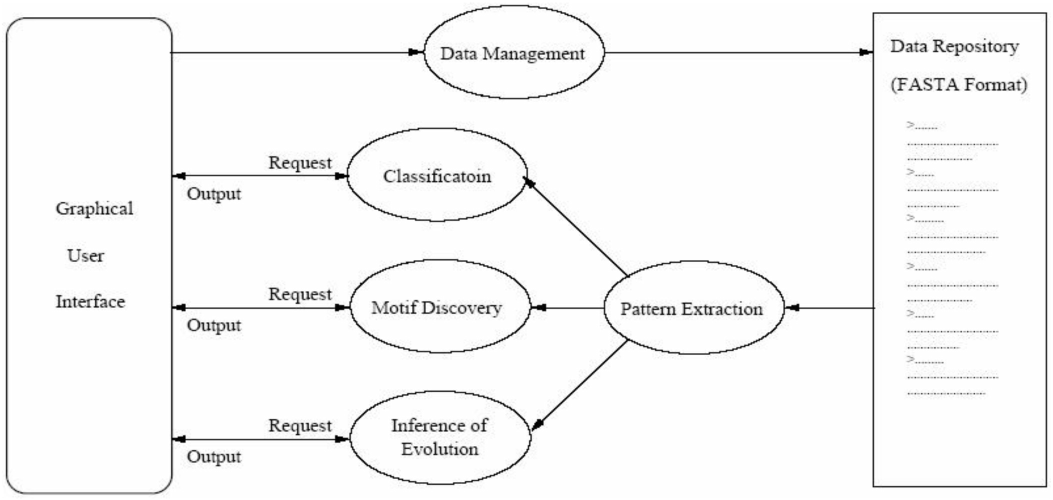


Figure 1. Overview of GAPE

The flowchart of the pattern extraction algorithm is shown in Figure 2, which consists of two major components: *genetic algorithm processing* and *pattern construction*. The genetic algorithm processing component is shown in the central portion of the flowchart. Developing this component involves three parts: 1) design of an initial population construction mechanism, 2) design of applicable fitness functions, and 3) design of three GA operators for offspring generation. Each of these parts is carefully designed to satisfy our specific domain of problems. The *pattern construction* component as shown in the right hand portion of the flowchart is needed because optimal solutions obtained by the GA are strings containing only common elements found the sequences. It may lack the power to express elements where base residues vary in different sequences. Therefore, the GA-generated string may need further processes in order to incorporate information gathered from the original nucleic acid sequences regarding different variations of certain elements.

OPTIMIZED GENETIC ALGORITHM

The GA is a model of machine learning which derives its behavior from a metaphor of the processes of evolution in nature. This is done by the creation within a machine of a *population* of *individual*s. The individuals in the population then go through a process of evolution: use GA *operator*s to generate offspring of a given population, and then use a *fitness function* to select the more fitted individuals to form the next generation of population. This evolution process repeats till the optimal solution is found or no further improvements can be made.

The optimized GA used in our algorithm follows the same technique as described above. In this section we emphasize the three major efforts in the design of the optimized GA for nucleic acid pattern extraction: 1) each *individual* in the population is represented as a string of residues, which is used to express the common parts found in the given nucleic acid sequences. In order to carry out these common parts information, initial population is constructed based on a fast cluster analysis procedure; 2) *fitness functions* are designed to reflect all sequences in the database; 3) GA *operators* are redesigned because "too much" change may be made to the individuals if standard operators were performed in our specific domain of problems. These changes are explained in detail in the rest of this section.

Construction of the initial population

Construction of an initial population is an important procedure in GA mainly because all offspring are depended on it. As a result, a *better* initial population generally yields a faster termination. However, the term *better* is loosely defined and varies depending on the domain of the problem. Another challenge is that the initial population should be generated in a timely fashion in order not to cumber the overall efficiency.

The initial population construction employed in our optimized GA uses a fast statistical analysis indicated as "cluster analysis" in the flowchart. The idea of using cluster analysis is borrowed from a multiple sequence alignment tool named ClustalW (Thompson and Gibson et al., 1994), though different measures are used in our implementation. The procedure binds more identical sequences in the database together, which are then used to form strings that contain their common parts. Thus more common elements may be collected in our initial population by using this heuristic.

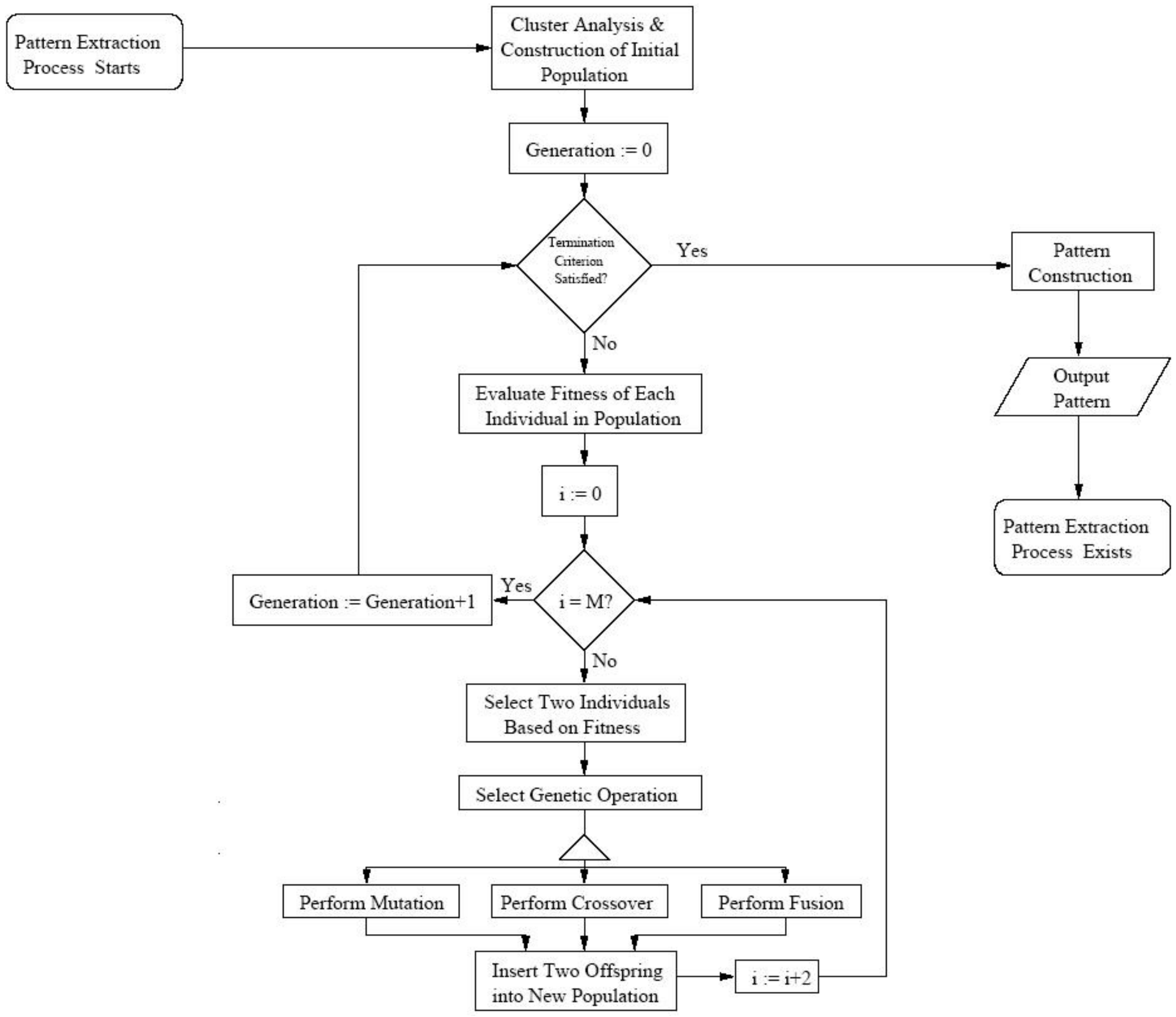
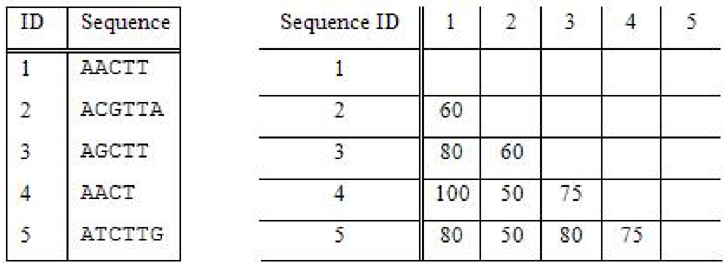

Pattern Extraction Process Starts
Cluster Analysis & Construction of Initial Population
Generation := 0
Termination Criterion Satisfied?
Yes
No
Pattern Construction
Output Pattern
Pattern Extraction Process Exists
Evaluate Fitness of Each Individual in Population
i := 0
i = M?
Yes
No
Generation := Generation+1
Select Two Individuals Based on Fitness
Select Genetic Operation
Perform Mutation
Perform Crossover
Perform Fusion
Insert Two Offspring into New Population
i := i+2

Figure 2. Flowchart of pattern extraction algorithm

Table 1 illustrates the cluster analysis procedure. A matrix is used to record the percentage identity of each DNA comparing against another in the database. Percentage identity of two sequences is computed as follows:

$$\text{Percentage Identity} = \frac{\text{Number of Identical Residues} \times 100}{\text{Length of the Smaller Sequence}}$$

Although a more sophisticated measures may be used, they may require much more time hence reduce the overall efficiency of the algorithm. The following table shows the fully computed percentage identity table of the 5 sequences given on the left hand side in Table 1. Since the percentage identity matrix is symmetric, only the lower half needs to be computed. To compute the percentage identity of sequence 1 and 2, we first need to determine the length of the smaller sequence: it is the length of sequence 1, i.e., 5. The next step is to get the number of identical residues: positions 1, 4 and 5 are the only places where sequence 1 and 2 has identical residues, thus the number is 3. By plug in these values into the above formula, we can obtain the percentage identity of sequence 1 and 2, which is $3 \times 100 / 5 = 60$. Similarly, the rest of the table can be computed.



ID	Sequence
1	AACTT
2	ACGTTA
3	AGCTT
4	AACT
5	ATCTTG

Sequence ID	1	2	3	4	5
1					
2	60				
3	80	60			
4	100	50	75		
5	80	50	80	75	

Table 1. Cluster analysis – percentage identity score

Once the percentage identity table has been constructed, the initial population construction procedure scans through the matrix and searches for the two nucleic acid sequences that produced the highest score in the current matrix (break ties randomly). *Longest common subsequence* of the two selected sequences is then obtained and stored into the initial population. The corresponding score of these two sequences in the percentage identity table is then set back to 0; hence they will not be re-selected again as a pair. They are, however, still eligible to be paired up with other sequences depending on their corresponding scores. This process repeats until all individuals in the initial population have been initialized.

Fitness function of genetic algorithm

A good fitness function is an essential part of GA because it determines which individuals in the current population will be kept while the rest of them will be discarded. It varies depending on the problem and should reflect the properties of the subject to be optimized.

The fitness function employed by GAPE is *best average score*. Each individual in the population is pair-wise aligned with each sequence in the database and its average alignment score is recorded. The ones with higher average score will be assigned a higher value. This function is applied to guard each generation till the termination of GA. It guarantees that the newer generation is at least as good as the previous ones. After the current generation has been selected, then it is the responsibility of the GA operators to construct the next generation to be examined.

Genetic algorithm operators

GA operators are responsible for constructing the offspring based on a given population. The GA operators have to maintain the majority of the characteristics of the individuals in the current population while making some minor changes to see if they are better off using fitness function.

Three GA operators have been developed for the pattern extraction tool. These operators follow the essential idea of standard GA operators that better individuals (i.e., strings with higher scores assigned by the fitness function) have a greater chance to be selected to produce offspring. These three operators are discussed as follows.

Mutation operates on a single individual and generates one child. It randomly picks a position in the string and replaces the base character at this position with a different base character. This operator is mainly used to get around the local optimum.

Crossover operates on two individuals and generates two children. Different from standard crossover, our two chosen individuals may not have the same length. Furthermore, the operation must not destroy their common parts. Therefore, it is very important that pair-wise alignment of these two strings is performed first. The procedure then randomly picks two positions and interchanges the substring bounded inside these two positions. Finally the gaps created by alignment are removed to form the two children. Notice that performing pair-wise alignment prior to crossover ensures that the common elements obtained in the parents still remain in both children so that common residues in the current population are inherited.

The *fusion* operator also operates on two individuals, but only generates one child by returning the longest common subsequence of the two chosen individuals. The idea of this operation is to remove some of the noises in the selected strings to produce a more condensed individual containing purer common elements.

The following example illustrates these three GA operators. Each of these three operators has a probability associated with them so that some will be performed more often than the others during execution. The current assignment of the probabilities for *mutation*, *crossover* and *fusion* is 20%, 40% and 40%, correspondingly.

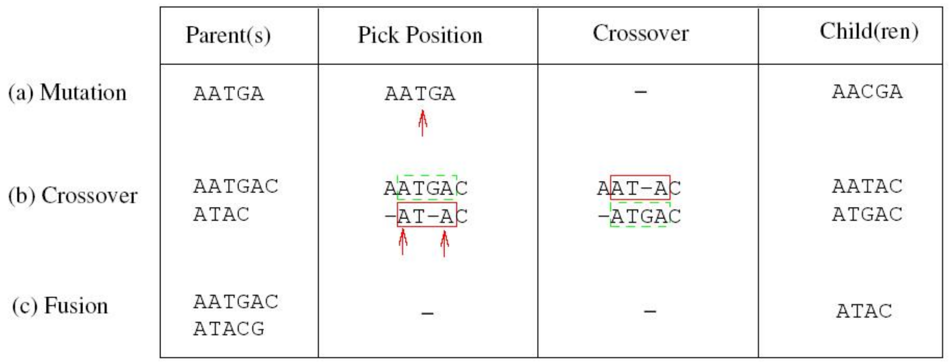


Figure 3. The three GA operators

This GA keeps searching for the optimal solution until *termination criterion* is satisfied. The termination criterion currently employed by the tool returns true if the best individual in the population remains the same for *N* generations. A larger *N* value yields more accurate result, but also takes longer time to terminate. The actual value of *N* may be determined by the users at their preferences.

PATTERN CONSTRUCTION ALGORITHM

Upon termination of GA, the next step of the process is to generate a meaningful pattern that has the power to express its family based on the string returned by GA. This process is called *pattern construction*. The GA-generated optimal string, denoted as *GA_String* for simplicity, carries common elements obtained from the nucleic acid sequences. These common elements, even though adjacent to each other in the *GA_String*, may appear apart in the original sequences, due to the uncommon parts in between the common elements. Therefore, a mechanism must be provided in order to incorporate the information carried in the original sequences into *GA_String* to form the final global pattern for this nucleic acid family.

We use *multiple sequence alignment* (MSA) to achieve this incorporation. MSA locates and aligns the common elements found in each record, which are later transformed into concrete base characters in the pattern. These common elements, represented as concrete characters in *GA_String*, may not appear exactly the same in each of the original sequences, i.e., some minor variations are allowed. However, these common elements are representatives of the majorities. As oppose to these concrete characters in *GA_String*, its gaps produced by MSA indicate the positions where base characters vary in different records and no dominating character could be obtained. Combining these two observations together, concrete characters and gaps in *GA_String* combine with original sequences will give us information needed to construct the desired global pattern. The MSA component used in our tool is not implemented from scratch; instead, a slightly modified version of the well developed MSA tool named ClustalW (Thompson and Gibson et al., 1994) is used. The most important modification of ClustalW is the cluster analysis procedure: in addition to group sequences based on their pair-wise alignment score only, the formation of the tree centers on the *GA_String*. Because the *GA_String* contains the common elements obtained from the family of sequences, this modified MSA algorithm determines the gap positions even more accurately.

The global pattern is defined using a modified regular expression. The modification can be described as follows: 1) only one regular expression operator, '`[]`', is employed to represent a character class; 2) probability (computed using percentage) information is incorporated into each character class, and is used to compute the closeness of a sequence comparing with the obtained pattern. E.g., '`[A(.7)T(.1)-(.2)]`' implies that the probability of 'A' appear at this particular position is 70%, where the probabilities for 'T' and '–' (gap) would be 10% and 20% respectively. Notice that we do allow gaps to be included into the character classes so that a sequence will not be reported dissimilar to the pattern solely because they are shorter. This is to say that we focus more on the fact that whether a sequence contains the common elements found in the family or not, rather than solely the length information. By incorporating percentage information into the modified regular expression, more precise patterns can be generated. The global pattern generation process is formally presented below:

(1) Perform MSA of *GA_String* with the original sequences;
(2) Scans through the aligned *GA_String*:
  a. **If** the current position is a base character, map it to the same base character;
  b. **Otherwise**, map the current position to a character class containing all possible characters (including gap) occurred in the current *column*, associated with their percentage information.

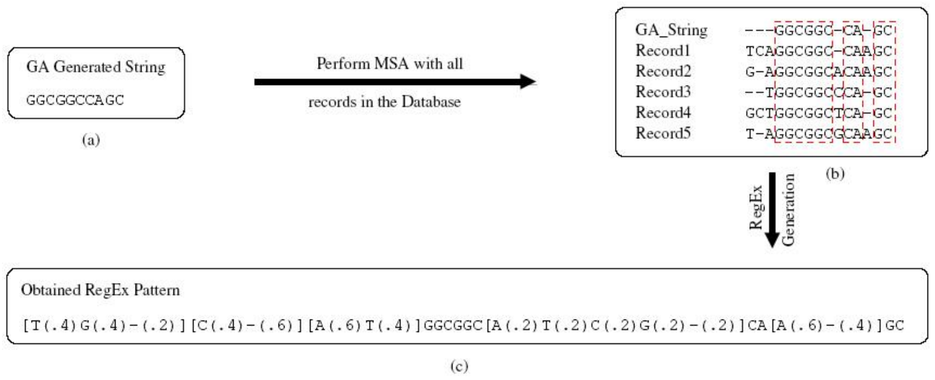


Figure 4. Output construction process for a simple pattern extraction problem

Figure 4 illustrates pattern generation process: Figure 4(a) shows the *GA_String*; The incorporation is done by performing MSA as shown in Figure 4(b). Here, the first column contains T, G and gap '-'. The percentage information for T is 2 (occurrences) out of 5 (total records), i.e., 2/5 = 0.4. Similarly, we can calculate this information for G and gap, which are 0.4 and 0.2 respectively. Therefore, the first position is mapped to character class `[T(.4)G(.4)-(.2)]`. This procedure continues on to the second column, and it is mapped to `[C(.4)-(.6)]`. After scanning through the entire *GA_String*, the optimized regular expression pattern is obtained as shown in Figure 4(c). At this point, finally, the desired global pattern of a family of given nucleic acid sequences is successfully obtained.

CLASSIFICATION USING CLOBAL PATTERN

We first introduce a specialized pair-wise alignment algorithm method which is used to measure the closeness of a given sequence with an obtained global pattern. The algorithm aligns a nucleic acid sequence with an obtained pattern represented in the form of regular expression, using dynamic programming techniques. The alignment algorithm is similar to Needleman-Wunsch algorithm (Needleman and Wunsch, 1970), with two modifications: the gap penalty and matching information is incorporated into the scoring matrix; and probabilities of each residue appearing at a particular position are taken into consideration when computing the alignment scores.

The scoring matrix used in Needleman-Wunsch algorithm contains values for every possible residue match. We have extended this matrix to incorporate gap penalty information as well since gaps may be included in the character classes in the specialized pair-wise alignment. The following is an example of such an extended matrix:

	A	T	C	G	-
A	4				
T	0	9			
C	0	-3	6		
G	0	-1	-2	5	
-	-GapPenalty	-GapPenalty	-GapPenalty	-GapPenalty	(4+9+6+5)/4

Table 2. Specialized scoring matrix

In the extended scoring matrix, a gap (represented using '-') is treated as a residue. The scores for matching any non-gap residue with a gap is "-GapPenalty". This is consistent with subtracting the GapPenalty in Needleman-Wunsch algorithm. The score for a *gap-to-gap* matching is needed so that the short nucleic acid sequences will not get low score due to their shortness. For example, when aligning a sequence *S* with a pattern *P*, if a character class in *P* at a particular position contains gap, it implies that a gap is allowed at the same position in *S*. This score is computed by taking the average of the exact matching scores (scores on the diagonal).

The specialized alignment algorithm needs to take the probabilities of each residue in a character class into consideration. Thus, we need to modify the alignment score computation procedure. In our modified regular expression, a concrete character, e.g. "`C`", may be expressed as "`[C(1)]`". Therefore, each character class (e.g. "`[C(.4)-(.6)]`") and concrete character (e.g. "`[C(1)]`") in the obtained pattern can be considered as a single *token*. Each token contains the possible characters allowed at a particular position along with their probabilities. Thus, given a nucleic acid sequence *S* and a global pattern P, where $S = s_1s_2 \ldots s_m$ (s_i denotes a base character) and $P = p_1p_2 \ldots p_n$ (p_i denotes a token), the alignment score of S can be found by successively finding the best score for aligning $s_1, s_2, \ldots, s_i$ with $p_1, p_2, \ldots, p_j$, $\forall i \in [1, m]$ and $\forall j \in [1, n]$. The alignment score is calculated through a matrix M where each element of M is calculated as follows:

$$M_{i,j} = \max\begin{cases} M_{i-1,j-1} + score(s_i, p_j) & (1) \\ M_{i,j-1} + score('-', p_j) & (2) \\ M_{i-1,j} + score(s_i, '-') & (3) \end{cases}$$

where $score(x, y)$ is obtained by $score(x, y) = \sum_{\forall \alpha \in \Re} (scoring\_matrix(x, \alpha) \times prob(\alpha, y))$ and R is the set of residues as well as '-'. E.g., in case of DNA sequences, R = {'A', 'T', 'C', 'G', '-'}. *Scoring_matrix*(x, α) retrieves the score from the scoring matrix for matching residue x with α; *prob*(α, y) is a short-hand notation used to express the probability of occurrence of α in token y. Following this computation, the alignment score of S and P is obtained at $M_{m,n}$. Intuitively, Equation (1) computes the alignment score by matching the residue at position i in S with the token at position j in P. Equation (2) computes the score when a gap is inserted at position i in S, and (3) computes the score when a gap is inserted at position j in P. Notice that in (2), if the probability of '-' in the current token is 0, i.e., no gaps are allowed at this position, then the full gap penalty will be taken. Similarly, the token is set to `[-(1)]` in (3) at all times, thus, the full gap penalty will always be taken if we try to insert a gap into the pattern during the procedure. In case a pattern contains only concrete characters, this alignment algorithm behaves the same as Needleman-Wunsch algorithm. Therefore, we can also understand this specialized alignment algorithm as the generalization of Needleman-Wunsch algorithm to the regular expression.

This specialized alignment score has the same property as the traditional one, that is, the more similar S is comparing to P, the higher score S will obtain. Therefore, all existing methods for *pattern matching* are still applicable using the new algorithm, and classification model can be constructed base on the global pattern and the scoring information.

To construct a classification model, the specialized alignment algorithm is performed on the final pattern with each sequence in the training data file. The average alignment score and standard deviation is recorded. In fact, the classification model consists of the pattern obtained, the average alignment score *avg*(score), and the standard deviation σ. Once such a model has been constructed, a new sequence may be submitted for classification. The tool aligns the input sequence with the pattern and records its alignment score. Estimation of membership is made based on how far away this alignment score is from the average in terms of σ. If the difference is within σ, the input sequence is highly likely to be a member of the family; on the other hand, if the difference is greater than $2 \times \sigma$, we reject the sequence stating that it is probably not a member. Therefore, a final score of a sequence denoted as the *GAPE score* is introduced. GAPE score of a sequence S is essentially the alignment score obtained by aligning S with the pattern, and then subtracted by the lower cutoff line. This is shown using formula (4) below:

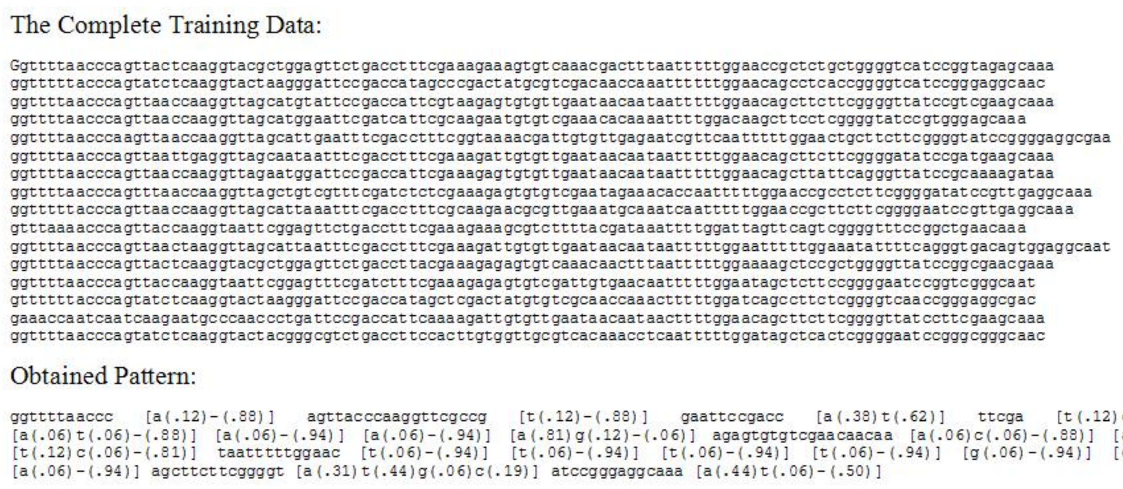
$$\text{GAPE Score} = \text{Alignment Score} - (avg(\text{score}) - 2 \times \sigma) \qquad (4)$$

Thus, sequences with non-negative GAPE scores are considered as members of the family; while sequences with negative GAPE scores are considered not to be members.

EXPERIMENTAL RESULTS

The GAPE tool has been tested using real biological databases. The data used for testing is a part of Rfam obtained from (Jones and Bateman, et al., 2003). We have tested multiple RNA families for pattern extraction and classification, and the results are presented in this section.

The RNA sequences listed below is an RNA family example obtained from Rfam. The obtained pattern does carry the common elements found in the training data. After performing MSA (results not shown here), on the positions where residues vary too much for them to be considered as common elements, character classes are captured during pattern construction procedure.



The Complete Training Data:

```
Ggttttaacccagttactcaaggtacgctggagttctgacctttcgaaagaaagtgtcaaacgactttaattttttggaaccgctctgctggggtcatccggtagagcaaa
ggtttttacccagtatctcaaggtactaagggattccgaccatagcccgactatgcgtcgacaaccaaatttttttggaacagcctcaccggggtcatccgggaggcaac
ggttttaacccagttaaccaaggttagcatgtattccgaccattcgtaagagtgtgttgaataacaataattttttggaacagcttcttcggggttatccgtcgaagcaaa
ggttttaacccagttaaccaaggttagcatggaattcgatcattcgcaagaatgtgtcgaaacacaaaattttggacaagcttcctcggggtatccgtgggagcaaa
ggttttaacccaagttaaccaaggttagcattgaatttcgacctttcggtaaaacgattgtgttgagaatcgttcaatttttggaactgcttcttcggggtatccggggaggcgaa
ggttttaacccagttaattgaggttagcaataatttcgacctttcgaaagattgtgttgaataacaataattttttggaacagcttcttcggggatatccgatgaagcaaa
ggttttaacccagttaaccaaggttagaatggattccgaccattcgaaagagtgtgttgaataacaataattttttggaacagcttattcagggttatccgcaaaagataa
ggttttaacccagtttaaccaaggttagctgtcgtttcgatctctcgaaagagtgtgtcgaatagaaacaccaatttttggaaccgcctcttcggggatatccgttgaggcaaa
ggtttttacccagttaaccaaggttagcattaaatttcgacctttcgcaagaacgcgttgaaatgcaaatcaatttttggaaccgcttcttcggggaatccgttgaggcaaa
gtttaaaacccagttaccaaggtaattcggagttctgacctttcgaaagaaagcgtcttttacgataaattttggattagttcagtcggggtttccggctgaacaaa
ggttttaacccagttaactaaggttagcattaatttcgacctttcgaaagattgtgttgaataacaataattttttggaatttttggaaatattttcagggtgacagtggaggcaat
ggttttaacccagttactcaaggtacgctggagttctgaccttacgaaagagagtgtcaaacaactttaattttttggaaaagctccgctggggttatccggcgaacgaaa
ggttttaacccagttaccaaggtaattcggagtttcgatctttcgaaagagagtgtcgattgtgaacaattttttggaatagctcttccggggaatccggtcgggcaat
gtttttttacccagtatctcaaggtactaagggattccgaccatagctcgactatgtgtcgcaaccaaacttttttggatcagccttctcggggtcaaccgggaggcgac
gaaaccaatcaatcaagaatgcccaaccctgattccgaccattcaaaagattgtgttgaataacaataactttttggaacagcttcttcggggttatccttcgaagcaaa
ggttttaacccagtatctcaaggtactacgggcgtctgaccttccacttgtggttgcgtcacaaacctcaattttttggatagctcactcggggaatccgggcgggcaac
```

Obtained Pattern:

```
ggttttaaccc   [a(.12)-(.88)]   agttacccaaggttcgccg   [t(.12)-(.88)]   gaattccgacc   [a(.38)t(.62)]   ttcga   [t(.12)c(.06)-(.81)]
[a(.06)t(.06)-(.88)]  [a(.06)-(.94)]  [a(.06)-(.94)]  [a(.81)g(.12)-(.06)]  agagtgtgtcgaacaacaa  [a(.06)c(.06)-(.88)]  [a(.06)-(.94)]
[t(.12)c(.06)-(.81)]  taatttttggaac  [t(.06)-(.94)]  [t(.06)-(.94)]  [t(.06)-(.94)]  [t(.06)-(.94)]  [g(.06)-(.94)]  [g(.06)-(.94)]
[a(.06)-(.94)] agcttcttcggggt [a(.31)t(.44)g(.06)c(.19)] atccgggaggcaaa [a(.44)t(.06)-(.50)]
```

The effectiveness of the classification method is also examined. Figure 5 illustrates the classification results performed on four different Rfam families. In each figure, circles are used to denote the sequences within the same families as the training data, and crosses are used to represent other sequences outside the testing families. Except a very few outliers, all sequences within the families of the testing data scored above 0, where as sequences from other classes obtained much lower scores. Notice that there are clear indications even for the outliers that have scored below 0 as well (they are still much higher than scores obtained by sequences from outside the family). Total of eight randomly selected Rfam families are tested to investigate the effectiveness of GAPE. The total number of misclassified sequences for these eight families was 14 out of 3041, i.e., the overall accuracy for GAPE is 99.5%. All of these misclassified sequences are false negative.

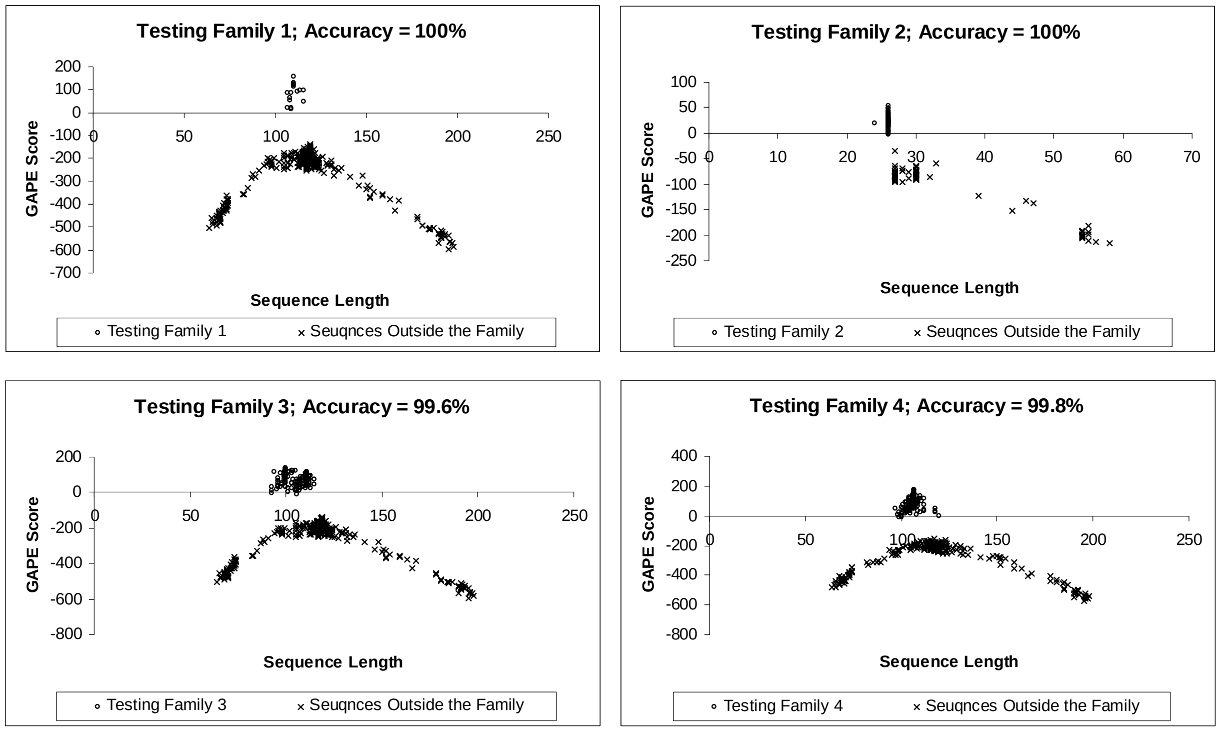


Figure 5. Classification results: GAPE Score≥ 0 $\Leftrightarrow$ member of the family

Our experimental results for both pattern extraction and classification indicate that: the pattern we obtained to express a nucleic acid family can be used for classification with satisfactory results; and more importantly, the pattern extraction algorithm presented in this paper is indeed effective.

CONCLUSION

We introduced a tool taking an optimized genetic algorithm based approach to extract global nucleic acid patterns effectively. The optimized GA algorithm simulates an evolutionary process to optimize the strings containing the pattern information of a nucleic acid family during the process. This process ensures the obtained string contains the most common elements among these sequences. Major optimizations done on the design of the GA can be summarized as follows: the initial population of GA is generated through an efficient *cluster analysis* procedure in order to increase accuracy; and optimized genetic operators named *mutation*, *crossover* and *fusion* take sufficient advantages of alignment results among nucleic acid sequences, and they ensure the common elements obtained in the current population are carried out to the next generation. The optimal string found by the optimized GA then goes through a pattern construction process to incorporate regular expression operators to enrich its expressive power.

Testing results using real biological data have shown that the patterns extracted by our tool are capable of representing their family members. In addition, a classification model can be constructed based on the obtained global pattern for a nucleic acid family. The classification models for multiple RNA families were verified using sequences from within the family and outside the family. Satisfactory results were obtained.

ACKNOWLEDGMENTS

We thank Professor William Tapprich in the department of biology and Professor Hesham Ali in the department of computer science for their helpful comments.

REFERENCES

1. Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990) Basic Local Alignment Search Tool. *J. Mol. Biol*, 215, 403-410.
2. Bailey, T. L. and Elkan, C. (1995) Unsupervised Learning of Multiple Motifs in Biopolymers Using Expectation Maximization. *Machine Learning*. 21(1-2), 51-80.
3. Brazma, A., Jonassen, I, Eidhammer, I., Gilbert, D. (1998) Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology,* Vol. 5, no. 2, 277-304.
4. Buhler, J. and Tompa, M. (2002) Finding Motifs Using Random Projections. *J. Comput. Biol.*, 9(2), 225-242.
5. Conklin, D. and Farrah, T. (1998) Protein motif recognition in DNA sequences containing indel errors. *German Conference on Bioinformatics 1998*.
6. Eddy, S. R. (1998) Profile Hidden Markov Models. *Bioinformatics*, 14, 755-763.
7. Hertz, G. and Stormo, G. (1999) Identifying DNA and Protein Patterns with Statistically Significant Alignments of Multiple Sequences. *Bioinformatics*, 15, 563-577.
8. Jones, S., Bateman, A., Marshall, M., Khanna, A. and Eddy, S. (2003) Rfam: an RNA family database. *Nucl. Acids Res.*, 31, 1, 439-441.
9. Lawrence, C., Altschul, S., Boguski, M., Liu, J., Neuwald, A. and Wootton, J. (1993) Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment. *Science*, 262, 208-214.
10. Needleman, S. B. and Wunsch, C. D. (1970) Needleman-Wunsch Algorithm for Sequence Similarity Searches. *Biol.* 48, 443-453.
11. Pevzner, P. A. and Sze, S. H. (2000) Combinatorial Approaches to Finding Subtle Signals in DNA Sequences. *8th Int. Conf. on Intell. Sys. for Mol. Biol.*, 269-278.
12. Quest, D. and Ali, H. (2004) Ontology Specific Data Mining Based on Dynamic Grammars. *CSB2004*, 495-496.
13. Raphael, B., Liu, L., Varghese, G. (2004) A Uniform Projection Method for Motif Discovery in DNA Sequences. *TCBB*, Vol. 1, Issue 2, 91-94.
14. Rigoutsos, I. and Floratos, A. (1998) Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. *Bioinformatics*, Vol. 14, No. 1, 56-67.
15. Thompson, J., Gibson, T. and Higgins, D., (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22, 4673-4680.