

Chapter 1

Pattern Discovery and Classification in Biosequences

Jason T. L. Wang, Thomas G. Marr, Steve Rozen,
Dennis Shasha, Bruce A. Shapiro, Gung-Wei Chirn,
Zhiyuan Wang and Kaizhong Zhang

Chapter Summary

Pattern discovery and classification are two important problems in biosequence processing. This chapter presents one case study for each problem. We start by describing a method for finding active patterns in a set of sequences. The method is an automatic two step process: (1) find candidate patterns in a small sample of the sequences; (2) test whether these patterns are approximately present in all the sequences. To reduce the running time, we develop two optimization heuristics based on statistical estimation and pattern matching techniques. Experimental results on both protein and synthetic data show the effectiveness of the heuristics. We then apply the pattern discovery method, together with a fingerprint technique, to conducting DNA classification. Empirical study shows a promising result when using our algorithms to classify Alu sequences.

1.1 Introduction

With the significant growth of the size of biosequence data, it becomes increasingly important to develop new techniques for finding “knowledge” from the data. *Pattern discovery* is a fundamental operation in such applications. It attempts to find useful patterns in biosequences which can help scientists to analyze the property of a sequence or predict the function of a new entity. The discovered patterns may also help to *classify* an unknown sequence, i.e., assign the sequence to an existing family. In this chapter, we show how to discover active patterns in a set of protein sequences and classify an unlabeled DNA sequence. We use protein sequences as an example to illustrate our discovery algorithm, though the algorithm applies to sequences of any sort, including both protein and DNA.

1.2 Pattern Discovery in Protein Sequences

1.2.1 Preliminaries

The patterns we wish to discover within a set of sequences are regular expressions of the form $*X_1 * X_2 * \dots$. The X_1, X_2, \dots are *segments* of a sequence, i.e., subsequences made up of consecutive letters, and $*$ represents a variable length don't care (VLDC). In matching the expression $*X_1 * X_2 * \dots$ with a sequence S , the VLDCs may substitute for zero or more letters in S at zero cost.

The dissimilarity measure used in comparing two sequences is the *edit distance*, i.e., the minimum cost of edit operations used to transform one subsequence to the other after an optimal and zero-cost substitution for the VLDCs, where the edit operations include insertion, deletion and change of one letter to another [Wagner and Fischer, 1974; Zhang *et al.*, 1994]. That is, we find a one-to-one onto mapping from each VLDC to a subsequence of the data sequence and from each pattern subsequence to a subsequence of the data sequence such that (i) the mapping preserves the left-to-right ordering (if a VLDC at position i in the pattern maps to a subsequence starting at position i_1 and ending at position i_2 in the data sequence, and a VLDC at position j in the pattern maps to a subsequence starting at position j_1 and ending at position j_2 in the data sequence, and $i < j$, then $i_2 < j_2$); and (ii) the mapping has the minimum cost among all such mappings, where the cost of a mapping is the sum of the costs from pattern subsequences to data subsequences in the mapping. The edit distance is a useful measure of

S_1 : YDPMIEDKEYSRLVG
 S_2 : RMKQLGRTYDPAVWG
 S_3 : YDPMNWNFEKETLVG

Figure 1.1: The set \mathcal{S} of three sequences.

evolutionary distance [Sankoff and Kruskal, 1983]. For the purpose of this work, we assume that all the edit operations have a unit cost, though the techniques we propose do not depend on that cost assumption or essentially on the edit distance metric.

Example 4.1

Consider the set \mathcal{S} of three sequences in Figure 1.1. Suppose only exactly coinciding segments occurring in at least two sequences and having lengths greater than 3 are considered as ‘active.’ Then \mathcal{S} contains one active pattern:

$$*S_1[1, 4]* = *YDPM* \iff *S_3[1, 4]* = *YDPM*$$

where $V[x, y]$ is a segment of a sequence V from the x^{th} to the y^{th} letter inclusively. If patterns occurring in all the three sequences within distance one are considered as active, i.e., one mutation (mismatch, insertion or deletion) is allowed in matching a pattern with a sequence, then \mathcal{S} contains three active patterns:

$$\begin{aligned} & *S_1[1, 4]* = *YDPM* \\ \iff & *S_2[8, 11]* = *TYDP* \\ \iff & *S_2[9, 12]* = *YDPA* \\ \iff & *S_3[1, 4]* = *YDPM* \end{aligned}$$

If patterns having the form $*X * Y*$ are sought with lengths greater than 7 and one mutation allowed, then S_1 and S_3 share the following four active patterns:

$$\begin{aligned} & *S_1[1, 4] * S_1[12, 15]* = *YDPM * RLVG * \\ \iff & *S_1[1, 5] * S_1[13, 15]* = *YDPMI * LVG * \end{aligned}$$

$$\begin{aligned} \iff *S_3[1, 4] * S_3[12, 15]* &= *YDPM * TLVG * \\ \iff *S_3[1, 5] * S_3[13, 15]* &= *YDPMN * LVG* \end{aligned}$$

End of Example

Formally, let \mathcal{S} be a set of sequences. We define the occurrence number (or the activity) of a pattern to be the number of sequences in \mathcal{S} that match the pattern within the allowed distance. We say the occurrence number of a pattern P with respect to distance i and set \mathcal{S} , denoted $occurrence_no_{\mathcal{S}}^i(P)$, is k if $*P*$ matches k sequences in \mathcal{S} within distance at most i , i.e., the k sequences contain P within distance i . For example, in Figure 1.1, $occurrence_no_{\mathcal{S}}^0(*YDPM*) = 2$ and $occurrence_no_{\mathcal{S}}^1(*YDPM*) = occurrence_no_{\mathcal{S}}^1(*TYDP*) = occurrence_no_{\mathcal{S}}^1(*YDPA*) = 3$.

Given a set of sequences \mathcal{S} , we want to find all the active patterns P where P is within the allowed distance $Dist$ of at least $Occur$ sequences in \mathcal{S} and $|P| \geq Length$, where $|P|$ represents the number of the non-VLDC letters in the pattern P . ($Dist$, $Occur$, and $Length$ and the form of P are user-specified parameters.) The basic subroutine is to match a given pattern with a sequence in the given set. For example, in matching $*TQI*$ with a sequence $MYALTIHKR$, the first asterisk would substitute for $MYAL$ and the second asterisk would substitute for HKR . The distance is 1 (representing the cost of deleting Q). The length of the pattern $*TQI*$ is three.¹

To discover active patterns in a set of sequences, our overall strategy is first to find candidate segments among a small sample and then to combine the segments into candidate patterns. We then check which patterns satisfy the specified requirements. In the past, many techniques have been published to solve similar problems.² A commonly used one is based on multiple sequence alignment (see [Waterman, 1989] for review). The technique is useful when entire sequences in the set are similar. However, when the sequences have only short regions of local similarities, this approach makes no sense. There are also techniques based on local similarity search. The techniques work effectively when similarities meet some constraints, such as they occur in a predetermined number of sequences in the set [Roytberg, 1992], they differ by mismatches, but not by insertions/deletions [Bacon

¹Given a regular expression pattern P and sequence S , one can determine whether P is within distance $Dist$ of S in $O(Dist \times |S|)$ time when $O(|P|) = O(\log |S|)$ [Wu and Manber, 1992].

²These problems are mostly concerned with discovering patterns made up of single segments, or multiple segments separated by fixed length don't cares.

and Anderson, 1986], or they are situated at almost the same distance from the start of the sequences [Vingron and Argos, 1989]. In contrast to these techniques, our approach can find similarities composed of non-consecutive segments separated by variable length don't cares without prior knowledge of their structures, positions, or occurrence frequency.

1.2.2 Discovery Algorithm

Our algorithm consists of two phases: (1) find candidate segments among a small sample \mathcal{A} of the sequences; (2) combine the segments to form candidate patterns and evaluate the activity of the patterns in all of \mathcal{S} to determine which patterns satisfy the specified requirements.

Phase (1) consists of two subphases. In subphase A, we construct an index structure for the sequences in the sample. In subphase B, we traverse the structure to locate the candidate segments.

Subphase A of Phase 1

We construct a *generalized suffix tree* [Hui, 1992] (GST) for the sample of sequences. A suffix tree is a tree-like data structure that compactly represents a string by collapsing a series of nodes having one child to a single node whose parent edge is associated with a string [Landau and Vishkin, 1989; McCreight, 1976]. A GST is an extension of the suffix tree, designed for representing a set of strings. Each suffix of a string is represented by a leaf in the GST. Each leaf is associated with an index i . The edges are labeled with character strings such that the concatenation of the edge labels on the path from the root to the leaf with index i is a suffix of the i th string in the set. See Figure 4.2 for an example (the node labeled with a 1 above the leaf MTRM is an example of the result of a collapsing).³ The GST can be constructed asymptotically in $O(n)$ time and space where n is the total length of all sequences in the sample \mathcal{A} .

³Informally, the algorithm for constructing the GST works as follows. We append a unique symbol to each sequence in the sample and concatenate the sequences in the sample into a single one. We insert the suffixes of the sequences as into a trie except that if a node has only one child, we collapse the child with the parent and label the edge going down from the parent with a substring instead of a single character.

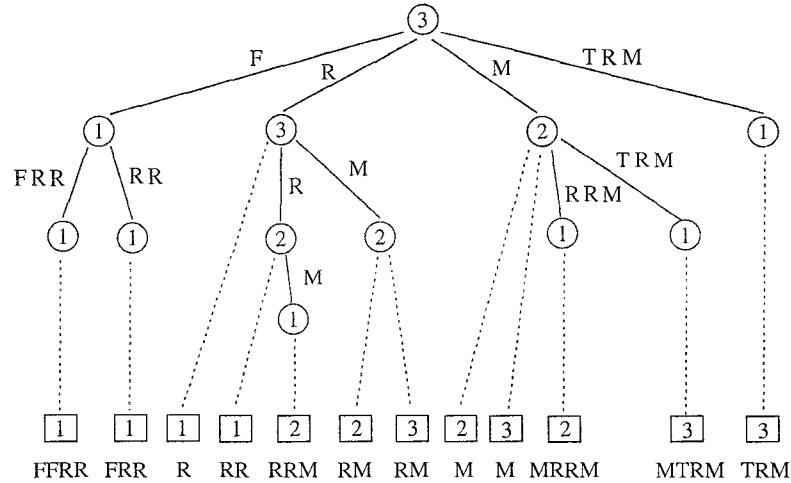


Figure 1.2: The GST for a sample $\mathcal{A} = \{\text{FFRR}, \text{MRRM}, \text{MTRM}\}$. Leaves are represented as rectangles, labeled with the indexes. Non-leaf nodes are represented as circles, labeled with the *count* values. The *count* value of a non-leaf node v represents the number of different indexes associated with the leaves in the subtree rooted at v . The suffix corresponding to a leaf is shown below the leaf. Note that the suffixes RM and M appear in two strings and hence appear twice in the leaves.

Subphase B of Phase 1

In subphase B, we traverse the GST constructed in subphase A to find all segments (i.e., all prefixes of strings labeled on root-to-leaf paths) that satisfy the length minimum. If the pattern specified by the user has the form $*X*$, then the length minimum is simply the specified minimum length of the pattern. If the pattern specified by the user has the form $*X_1 * X_2*$, we find all the segments V_1, V_2 where at least one of the V_i , $1 \leq i \leq 2$, is larger than or equal to half of the specified length and the sum of their lengths satisfies the length requirement. If the user-specified pattern has the form $*X_1 * X_2 * \dots * X_k*$, we find the segments V_1, V_2, \dots, V_k where at least one of the V_i , $1 \leq i \leq k$, is larger than or equal to $1/k$ th of the specified length and the sum of the lengths of all these segments satisfies the length requirement.

Phase 2

This phase also has two subphases. In subphase A, we evaluate the activity of the candidate patterns and rank them from highest to lowest according to their occurrence numbers in the sample with respect to distance $Dist$. If interesting patterns are of the form $*X_1 * X_2 * \dots$, we consider all possible combinations V_1, V_2, \dots of the segments obtained in phase (1) that meet the length requirement and match $*V_1 * V_2 * \dots$ with the sequences in the sample. Subphase B evaluates the most likely candidate patterns found in subphase A with respect to the entire set. (As our experimental results show later, screening out those unlikely candidate patterns in the first subphase saves significant time in the overall computation.)

1.2.3 Optimization Heuristics

In phase (2) of the discovery algorithm, we compare only the most likely candidate patterns with the entire set. The main question from an optimization point of view is which candidates to compare. Our strategy is as follows.

We use *simple random sampling without replacement* [Cochran, 1977] to select sample sequences from the set. Consider a candidate pattern P . Let M (a , respectively) denote the number of sequences in the entire set \mathcal{S} (the sample \mathcal{A} , respectively) that contain P within the allowed distance. Let N be the set size and n the sample size; $F = M/N$ and $f = a/n$. Then, with probability = 99%, F is in the interval (\hat{F}_L, \hat{F}_U) where

$$\hat{F}_L = f - \left(t \sqrt{\frac{N-n}{N-1}} \sqrt{\frac{f(1-f)}{n}} + \frac{1}{2n} \right),$$

$$\hat{F}_U = f + \left(t \sqrt{\frac{N-n}{N-1}} \sqrt{\frac{f(1-f)}{n}} + \frac{1}{2n} \right).$$

The symbol t is the value of the normal deviate corresponding to the desired confidence probability. When the probability = 99%, $t = 2.58$ [Cochran, 1977]. The values of N, n are given; f, a can be obtained from subphase A of phase (2). Thus, if the estimator $(\hat{F}_U \times N) < Occur$ for the candidate pattern P , then with probability $\geq 99\%$, P won't be an active pattern satisfying the specified requirements. We therefore discard it. This pruning will be referred to as *candidate pattern optimization*.

The second optimization heuristic we implemented is to eliminate the redundant calculation of occurrence numbers. Observe that the most expensive operation in our discovery algorithm is to find the occurrence number of a pattern with respect to the entire set, since that entails matching the pattern with all sequences. We say $*U_1 * \dots * U_m*$ is a *subpattern* of $*V_1 * \dots * V_m*$ if U_i is a subsegment of V_i , for $1 \leq i \leq m$.

One can observe that if P is a subpattern of P' , then $occurrence_no_S^k(P) \geq occurrence_no_S^k(P')$ for any distance parameter k . Thus, if P' is in the final output set, then we need not bother matching P with sequences in S , since it will be too. If P is not in the final output set, then P' won't be either, since its occurrence number will be even lower. We refer to this pruning strategy as *evaluation minimization*.

To illustrate how the above two optimization heuristics are incorporated into the discovery algorithm, consider finding the patterns of the form $*X * Y*$ whose total length is greater than or equal to 5. We begin by enumerating segments of length 3 in the generalized suffix tree (GST). Let $string(v)$ be the string on the edge labels from the root to v . If the above statistical estimator tells us that the combination of a segment $string(u_1)$ of length 3 with another segment $string(u_2)$ does not yield an active enough pattern satisfying the specified *Dist* and *Occur* requirements, then we eliminate the pair $string(v_1)$ and $string(v_2)$ from consideration, where v_1 and v_2 are descendants of u_1 and u_2 , respectively, in the GST. Similar pruning operations can be applied when enumerating longer segments in the GST.

1.2.4 Experiments and Results

We have carried out a series of experiments to evaluate the effectiveness and speed (measured by elapsed CPU time) of our approach. The data was a set of randomly generated 150 sequences, each having length 100. Every letter of the generated sequences was drawn randomly from the protein alphabet. For comparison purposes, we also tested the algorithms on real protein sequences. 150 proteins were selected randomly from the functionally related kinase family obtained from the Cold Spring Harbor Laboratory. The lengths of the kinase sequences ranged from 10 to 2938.

Table 1.1 shows the parameters and base values used in the experiments. The sequences in the sample were chosen randomly from the dataset. The parameter *NumSample* indicates the number of samples chosen for each dataset. In all the experiments presented here, only one sample was used in running a dataset. The sample size was obtained by multiplying *DSSize* by

SizeRatio. The patterns of interest had the form $*X*Y*$.

Parameter	Value	Description
<i>DSSize</i>	150	# of sequences in a dataset
<i>NumSample</i>	1	# of samples tested for a dataset
<i>SizeRatio</i>	20%	Ratio between sample size and dataset size
<i>Length</i>	5	Minimum length of an interesting pattern
<i>Dist</i>	1	Allowed distance between a pattern and a sequence

Table 1.1: Experimental parameters and base values used in performance analysis.

The metric used to evaluate the effectiveness of our algorithms is

$$HitRatio = \frac{NumDiscovered}{TotalNum} \times 100\%$$

where *NumDiscovered* is the number of interesting patterns discovered by our techniques. *HitRatio* stands for the percentage of the interesting patterns obtained from the exhaustive search method. The method works by considering all combinations of the segment pairs V_1, V_2 appearing in the dataset.⁴ One would like this percentage to be as high as possible.

Figure 1.3 shows the effectiveness of our approach for varying sample sizes. In this experiment, we had turned on both candidate pattern optimization and evaluation minimization when running our algorithms. The minimum occurrence number required *Occur* was set to 60 for the artificial data and was set to 8 for kinase. (The different parameter values were chosen to illustrate different results using different data.) Examining the graphs, we see that when $Dist = 0$ and $SizeRatio \geq 0.2$, our approach behaves almost like exhaustive search. When $Dist = 1$, the hit ratio reaches 80% provided the $SizeRatio \geq 0.4$. We were somewhat disappointed that smaller samples didn't give a better hit ratio, but research is like that sometimes.

We next compared the running times of the algorithms for the $Dist = 1$ case. Figure 1.4 shows the results. It can be seen that our algorithms run significantly faster than the brute force method. Even with $SizeRatio = 0.8$, in

⁴We have rejected approximately occurring patterns that never appear in the dataset yet satisfy the *Dist* and *Occur* constraints in favor of those that obey the constraints and do appear in the dataset. This is a theoretical limitation of our work that we have introduced to save computation time, though this also seems pragmatically to be a reasonable approach.

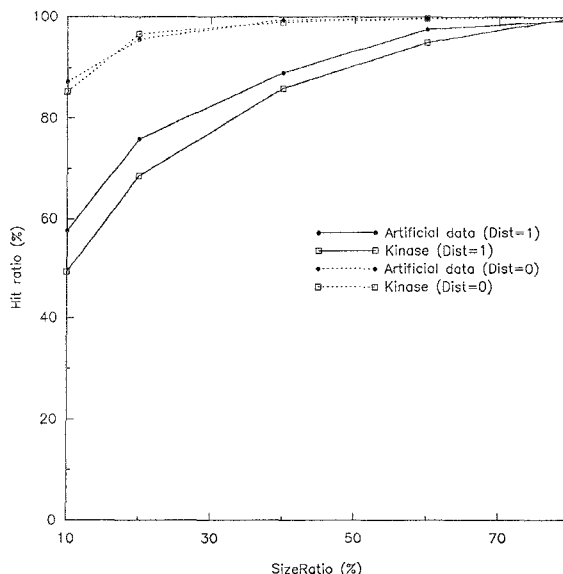


Figure 1.3: Effect of sample size.

which case the algorithms achieve nearly a 100% hit ratio, they are 10 times faster than exhaustive search. When the sample is this large, both segments V_1 , V_2 in an active pattern appear in the sample. Our algorithms work by enumerating all promising segment pairs in the sample, and therefore can find all the interesting patterns.

We also examined the effectiveness of the proposed optimization heuristics. To isolate the effect of the heuristics, we started by turning off the optimizations, and then turned on only one of them, and finally turned on both. To make the experiment manageable, we considered only patterns of the form $*X*$. The minimum occurrence number required $Occur$ was set to 55. The other parameters had the values shown in Table 1.1.

Figures 1.5 and 1.6 show the results obtained from the kinase sequences. (The results for the generated sequences are omitted since they lead to similar conclusions.) Examining the graphs, we see that very few active patterns were missed by the candidate pattern optimization. Pruning based on sub-pattern information works more effectively than that based on statistical estimation. Both optimizations together sped up the algorithms by a factor of nearly 100. We repeated the experiments by varying the compositions of samples and parameter values $Length$, $Dist$, $Occur$ with consistent results.

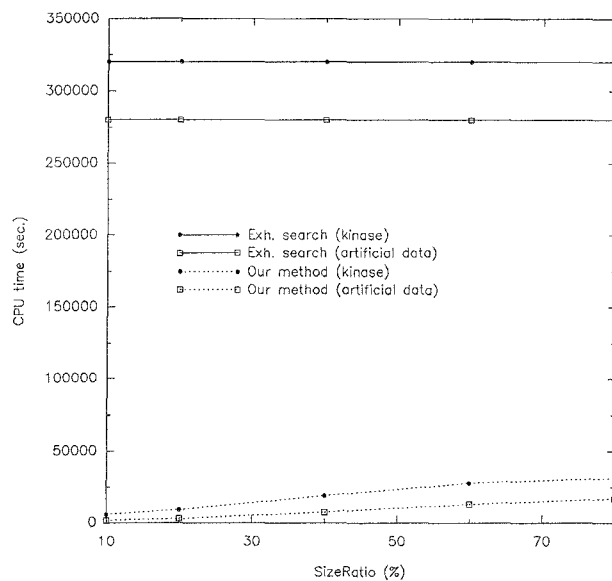


Figure 1.4: Comparison of running time.

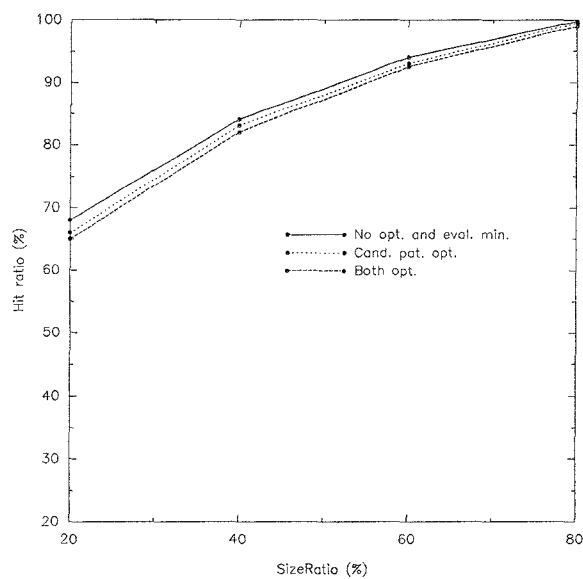


Figure 1.5: Performance of the pruning techniques.

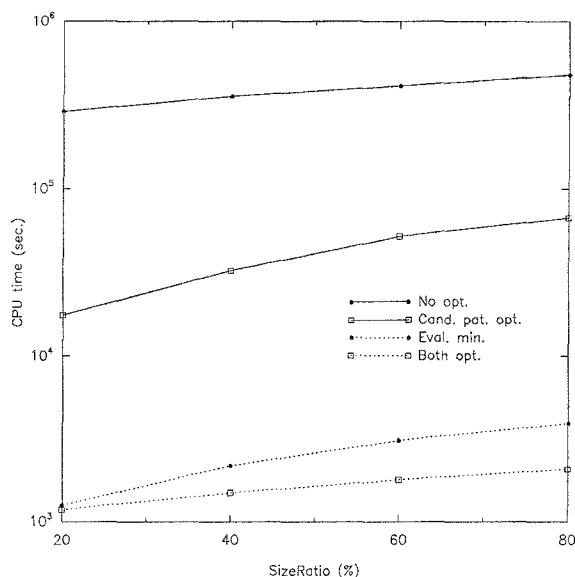


Figure 1.6: Efficiency of the pruning techniques.

To evaluate the usefulness of our pattern discovery algorithm, we have incorporated it into a classifier (referred to as PA) [Wang *et al.*, 1994b; Wang *et al.*, 1996a; Chirn, 1996] to classify 698 groups of related proteins in the SWISS-PROT protein sequence databank version 27 [Bairoch and Boeckmann, 1992]. These groups are categorized based on the documentation given in the PROSITE catalog version 11.0 [Bairoch, 1992]. The groups comprise more than 15,000 proteins.

Our classifier works by taking 70% of the proteins in each group, using them as the training data, and finding active patterns from them. It then classifies the remaining 30% test sequences by matching them against the active patterns found for each group. The classifier gives the highest rank to the group with the most patterns matching with a test sequence.

For comparison purposes, we also ran the current best classifier, referred to as HH [Henikoff and Henikoff, 1991], on the dataset. HH generates a set of blocks for each group, where a block comprises ungapped aligned regions extracted from the sequences in the group. To classify a test sequence, HH matches the sequence against all the blocks and displays a collection of groups, ranked based on their relevance to the sequence.

Table 1.2 summarizes the classification results. A test sequence was

classified correctly by HH (respectively, PA) if its group was ranked highest by HH (respectively, PA). The table also shows the results when the two classifiers agreed (i.e., the highest ranked group returned by both of them was the same) and disagreed on their rankings.

Classification results	Percentage of the test sequences
HH was correct	93.5%
PA was correct	92.3%
HH and PA agreed and both were correct	86.7%
HH and PA agreed and both were wrong	0.5%
HH and PA disagreed and HH was correct	7.3%
HH and PA disagreed and PA was correct	4.8%
HH and PA disagreed and both were wrong	0.7%

Table 1.2: Comparison between HH and PA. (Note: The last five percentages in the table add up to 100%.)

Thus if HH and PA agree, the classification has a high likelihood of being correct. Specifically, the correct agreed-upon classification divided by the total agreed-upon classification is $86.7\% / (86.7\% + 0.5\%) = 99.4\%$. On the other hand, if HH and PA disagree, then the likelihood that one is right is $(7.3\% + 4.8\%) / (7.3\% + 4.8\% + 0.7\%) = 94.5\%$. Thus the two classifiers give information that is complementary to each other. Using the two classifiers together, one can obtain high confidence classifications (if they agree) or suggest a new hypothesis (if they disagree).

In addition to the protein classification, our pattern discovery algorithm also helps in DNA classification. The next section presents one such example.

1.3 Classification of DNA Sequences

DNA sequence classification is an important problem in computational biology [Gelfand, 1995]. Given an unlabeled sequence S , a classifier makes predictions as to whether or not the sequence belongs to a particular class

\mathcal{C} . Many computer-assisted techniques have been proposed for constructing classifiers from a library of labeled sequences. In general, these techniques can be categorized into the following three classes:

- consensus search – this approach takes a collection of sequences of the class \mathcal{C} and generates a “consensus” sequence which is then used to identify sequences in uncharacterized DNA [Staden, 1984; Galas *et al.*, 1985; Mulligan and McClure, 1986; Berg and von Hippel, 1987; Studnicka, 1987; O’Neill and Chiafari, 1989; Gelfand, 1995];
- inductive learning/neural networks – this approach takes a set of sequences of the class \mathcal{C} and a set of sequences not in \mathcal{C} and then, based on these sequences and using learning techniques, derives a rule that determines whether the unlabeled sequence S belongs to \mathcal{C} or not [Quinqueton and Moreau, 1985; Sallantin *et al.*, 1985; Lukashin *et al.*, 1989; Lapedes *et al.*, 1990; Hirst and Sternberg, 1992; Shavlik *et al.*, 1992; Hirsh and Noordewier, 1994; Gelfand, 1995; Loewenstern *et al.*, 1995];
- sequence alignment – this approach aligns the unlabeled sequence S with members of \mathcal{C} using an existing tool such as FASTA [Lipman and Pearson, 1985; Pearson and Lipman, 1988] and assigns S to \mathcal{C} if the best alignment score for S is sufficiently high.

We review here two new methods, based on our pattern discovery algorithm and a hashing-based fingerprint technique, for calculating scores to classify DNA sequences. Our approach works by first randomly selecting a set \mathcal{B} of sequences of the class \mathcal{C} , referred to as “base data.” Then we take another set of sequences of \mathcal{C} , referred to as “positive training data,” and calculate, for each positive training sequence, a score with respect to the base sequences. The minimum score thus obtained is called the positive lower bound, denoted L_p . Next, we take a set of sequences not in \mathcal{C} , referred to as “negative training data,” and again calculate, for each negative training sequence, a score with respect to the base sequences. The maximum score thus obtained is called the negative upper bound, denoted U_n . Let $B_{high} = \max \{L_p, U_n\}$ and $B_{low} = \min \{L_p, U_n\}$ (see Figure 1.7). When classifying the unlabeled sequence S , we calculate S ’s score with respect to the base sequences, denoted c . If $c \geq B_{high}$, then S is classified to be a member of \mathcal{C} . If $c \leq B_{low}$, then S is classified not to be a member of \mathcal{C} . If $B_{low} < c < B_{high}$, then the “no opinion” verdict is given.

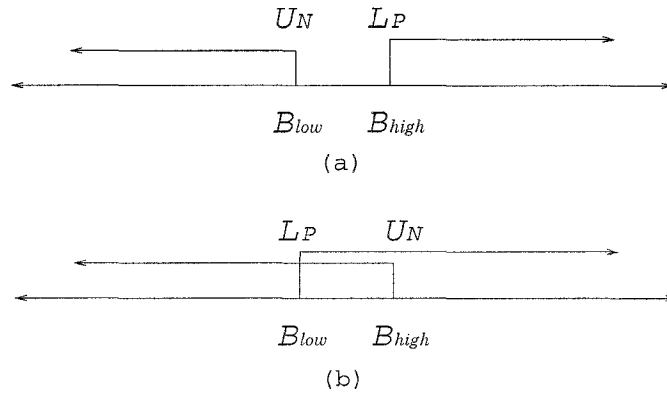


Figure 1.7: Illustration of the parameters B_{low} and B_{high} for cases (a) $U_N \leq L_P$ and (b) $U_N > L_P$.

The two proposed classifiers differ in their ways of processing the base sequences and calculating scores for the training and unlabeled sequences. We describe each classifier in turn in the following subsections.

1.3.1 Pattern-Based Classifier

The first classifier, referred to as the *pattern-based classifier*, applies the pattern discovery algorithm presented in Section 1.2.2 to find active patterns in the base data. Let \mathcal{R} be a set of active patterns discovered from the set \mathcal{B} of base sequences. Given a sequence S (which could be a training or an unlabeled sequence), the score between S and a pattern $P \in \mathcal{R}$, denoted $score(S, P)$, is defined as $|L|$ (i.e., the number of nucleotides in L), where L is the longest common substring of S and P . (The time complexity for finding the score is bounded by $O(|L|)$, and at worst $O(|S| \times |P|)$ [Clift *et al.*, 1986; Cobbs, 1994].) The score of S with respect to the base sequences is defined as

$$score(S) = \max\{score(S, P) | P \in \mathcal{R}\} \times 100.$$

1.3.2 Fingerprint-Based Classifier

The second classifier, referred to as the *fingerprint-based classifier*, adopts a hashing-based fingerprint technique to calculate scores [Califano and Rigoutsos, 1993; Wang *et al.*, 1996a]. Let S be a sequence and let Seg be a segment, i.e., a consecutive subsequence, of S . A gapped fingerprint f of Seg is a pos-

sibly non-contiguous subsequence of Seg that begins with the segment's first nucleotide. A gap at position p means that when forming f , we do not pick the nucleotide at position p in Seg . For example, let $S = \text{ACGTTGCA}$. Then $Seg = \text{ACGTTG}$ is a segment of S and ATT is a fingerprint of Seg with 2 gaps (one gap at position 2 and one gap at position 3). The number of gaps allowed in these fingerprints is bounded by a parameter gap .

Building Fingerprint Files

Given the set \mathcal{B} of base sequences, we pick the segments from each base sequence and hash each fingerprint of the segments into a file, as described below. Let S be a sequence in the base dataset \mathcal{B} . We take every segment Seg of length n from S and generate gapped fingerprints from Seg . The lengths of Seg 's fingerprints range from 2 to $n - 1$. Then we use a hash function h_k , $2 \leq k \leq n - 1$, to hash all fingerprints of length k to a fingerprint file \mathcal{F}_k . In the file, each fingerprint f is associated with a pair of integers (x, y) . This pair serves as the position marker for f , where x indicates that f is generated from a segment of the x^{th} sequence in \mathcal{B} and y means that the first nucleotide of f occurs at the y^{th} position in that sequence.

Example 4.2

Consider the following three base sequences: $S_1 = \text{ACGTTGCA}$, $S_2 = \text{ACCAGTGC}$, $S_3 = \text{CGGACTA}$. Suppose the length of segments is 6. Then, for example, we obtain the following segments from S_1 : ACGTTG , CGTTGC and GTTGCA .

Now consider the segment $Seg = \text{ACGTTG}$ taken from S_1 . Suppose $gap = 2$. Then, we can generate the following 3-nucleotide gapped fingerprints from Seg : ACG (0 gap); AGT (1 gap at position 2), ACT (1 gap at position 3); ATT (1 gap at position 2 and 1 gap at position 3), AGT (1 gap at position 2 and 1 gap at position 4) and ACT (1 gap at position 3 and 1 gap at position 4). Table 1.3 shows all gapped fingerprints generated from the segment ACGTTG .

Let $f = \text{XYZ}$ be a fingerprint of length 3. Suppose the hash function h_3 is $h_3(f) = (\text{num}(\text{X}) \times 4^2 + \text{num}(\text{Y}) \times 4^1 + \text{num}(\text{Z})) \bmod 7$, where $\text{num}(\text{X})$ is X 's ASCII value minus 64. Figure 1.8 shows the fingerprint file \mathcal{F}_3 for the three base sequences S_1 , S_2 and S_3 . Thus, for example, in bucket 1 in \mathcal{F}_3 , $\text{GGA}(3, 2)$ means that the fingerprint GGA is generated from S_3 and it starts from the 2^{nd} position in S_3 .

End of Example

	2-nucleotide fingerprints	3-nucleotide fingerprints	4-nucleotide fingerprints	5-nucleotide fingerprints
0 gap	AC	ACG	ACGT	ACGTT
1 gap	AG	ACT	ACGT AGT	ACGTG AGTTG
		AGT	ACTT	ACTTG
2 gaps	AT	ACT	ACGG ATTG	
		AGT	ACTG	
		ATT	AGTG	

Table 1.3: Gapped fingerprints (of lengths 2, 3, 4, 5, respectively) generated from the segment ACGTTG (the segment length is 6 and $gap = 2$).

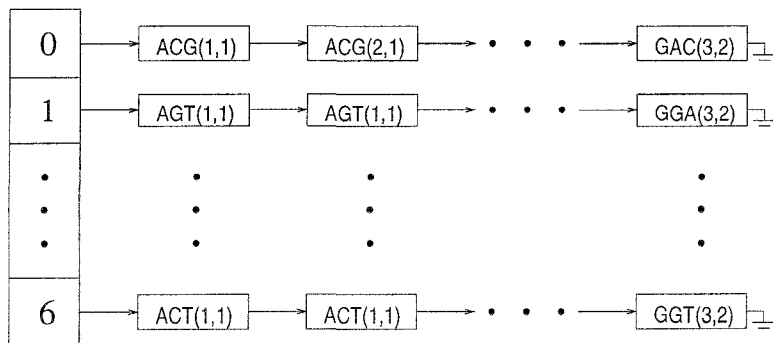


Figure 1.8: The fingerprint file \mathcal{F}_3 for the three base sequences in Example 4.2. Note that some entries are duplicate (e.g., AGT(1, 1) in bucket 1); they represent fingerprints with different numbers of gaps.

```

S1: ACGTTGCA
      |||
S:   CGATGCAT

```

Figure 1.9: Illustration of the alignment between S_1 and S .

Algorithm for Scoring

When calculating the score of a sequence S (whether it is a training or an unlabeled sequence), we segment S in the same way as for the base sequences and generate fingerprints from the resulting segments. We then hash the fingerprints, using the same hash functions as for the base sequences. When a match between S 's fingerprint and a base sequence's fingerprint occurs, we add one to the score of an appropriate position in the base sequence, as illustrated below.

Consider again the base sequence $S_1 = \text{ACGTTGCA}$ in Example 4.2 and a sequence $S = \text{CGATGCAT}$. Consider the 3-nucleotide fingerprint TGC starting from the 5th position in S_1 , and the same TGC starting from the 4th position in S . Let $q = 5$ and $p = 4$. We add one to the score of the position $q - p + 1 = 5 - 4 + 1 = 2$ in S_1 . Intuitively if we align the first nucleotide of S with the 2th nucleotide of S_1 , we can see a match between the two corresponding fingerprints (Figure 1.9). In general, if one aligns the first nucleotide of S with the k^{th} position in S_1 that obtains n scores in total, one can see n matches of fingerprints in the alignment. Thus, aligning the first nucleotide of S with the position in S_1 with the highest score may yield the best alignment between the two sequences. This technique was pioneered by Califano and Rigoutsos [1993] for finding the best alignment between two DNA sequences.

Figure 1.10 summarizes our scoring algorithm. The result is a histogram of scores on the base sequences. Figure 1.11 illustrates an example. Here we are given the sequence $S = \text{CGATGCAT}$ and the three base sequences in Example 4.2. The histogram is obtained after matching S 's fingerprints with all the fingerprints of the base sequences.

Let B be a base sequence in \mathcal{B} and let p be a position in B , $1 \leq p \leq |B|$. Let $\text{score}(B[p])$ represent the total scores added to the position p in B after applying the algorithm Scoring to the sequence S and the base sequences in \mathcal{B} . The score of B , denoted $\text{score}(B)$, is defined to be

$$\text{score}(B) = \max\{\text{score}(B[p]) \mid 1 \leq p \leq |B|\}.$$

The score of S with respect to the base sequences, denoted $\text{score}(S)$, is

Input: A sequence S , a set \mathcal{B} of base sequences and \mathcal{B} 's fingerprint files.
Output: A histogram of scores on the base sequences in \mathcal{B} .
 /* Let \mathcal{F} contain all fingerprints generated from S . */
for each fingerprint f in \mathcal{F} **do**
 begin
 /* Let the length of f be k . */
 hash f using h_k and probe into the fingerprint file \mathcal{F}_k ;
 for each match between f and a fingerprint \hat{f} in \mathcal{F}_k **do**
 begin
 /* Let the position marker associated with \hat{f} be (i, q) . */
 /* Suppose the first nucleotide of f occurs at the p^{th} position in S . */
 add one to the score of the position $q - p + 1$ in the i^{th} base
 sequence in \mathcal{B} ;
 end;
 end;
end;

Figure 1.10: Algorithm Scoring.

defined to be

$$score(S) = \frac{\max\{score(B) | B \in \mathcal{B}\}}{|S|} \times 100.$$

1.3.3 Experiments and Results

The algorithms for the proposed classifiers were implemented in C on a Sun SPARCstation 20 running the operating system Solaris version 2.4. We compared the relative performance of the algorithms by applying them to classifying Alu sequences [Jurka *et al.*, 1993; Claverie and Makalowski, 1994]. 327 Alu sequences were selected from the database in National Center for Biotechnology Information (NCBI) (ftp at ncbi.nlm.nih.gov/pub/jmc/alu/ALU.327.dna.ref). Among them, 100 were used as base sequences, 100 were used as positive training sequences, and the other 127 were treated as unlabeled test sequences. The lengths of these sequences ranged from 76 bp to 379 bp. In constructing non-alu data, following the studies in [Lukashin *et al.*, 1989; Demeler and Zhou, 1991], we randomly generated 1,200 sequences that retained the correct nucleotide frequencies using the simulation programming package SIMSCRIPT [Kiviat *et*

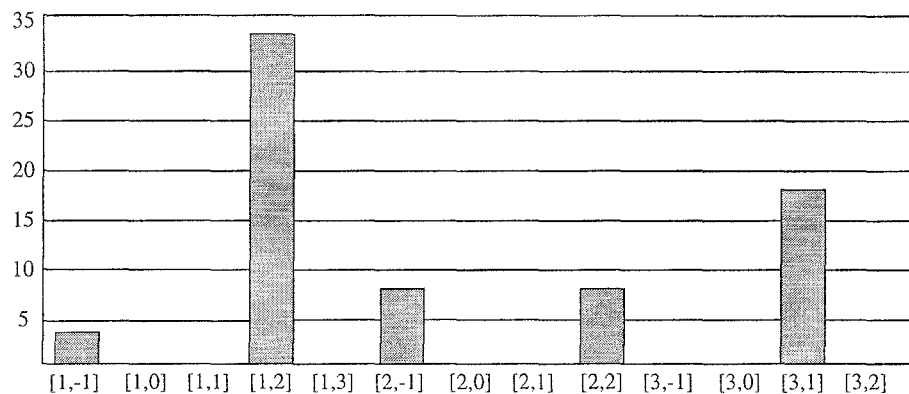


Figure 1.11: This histogram is obtained after processing the sequence $S = \text{CGATGCAT}$ and the three base sequences in Example 4.2. The y -axis shows scores. Each $[i, q]$ on the x -axis represents the q^{th} position in the i^{th} base sequence in Example 4.2.

al., 1983]. Among the data, 100 were used as negative training sequences and the other 1,100 were also treated as unlabeled test sequences. The lengths of these sequences ranged from 100 bp to 240 bp.

The pattern-based classifier found active patterns from the 100 base sequences. The active patterns had the form $*X*$, and had length greater than or equal to 11, occurrence number 15 and distance 0 (i.e., these patterns matched at least 15 base sequences without mutation). There were 556 active patterns, with lengths ranging from 11 bp to 22 bp. The fingerprint-based classifier fixed the segment length at 8 and *gap* at 0. Table 1.4 summarizes the parameters and their base values used in the experiments.

The metrics used to evaluate the effectiveness of our classification algorithms are precision rates (PR) and no-opinion rates (NR), where

$$PR = \frac{NumCorrect}{NumTest} \times 100\%$$

and

$$NR = \frac{NumNoOpinion}{NumTest} \times 100\%$$

$NumCorrect$ is the number of test sequences classified correctly, $NumNoOpinion$ is the number of test sequences obtaining the “no opinion” verdict, and

Parameter	Value	Description
$ \mathcal{B} $	100	Number of base sequences (Alu)
$ \mathcal{T}_P $	100	Number of positive training sequences (Alu)
$ \mathcal{T}_N $	100	Number of negative training sequences (non-Alu)
<i>NumTest</i>	1,227	Number of unlabeled test sequences (Alu & non-Alu)
<i>Length</i>	11	Minimum length of active patterns used in the pattern-based classifier
<i>Occur</i>	15	Minimum occurrence number of active patterns used in the pattern-based classifier
<i>Dist</i>	0	Allowed distance between an active pattern and a base sequence used in the pattern-based classifier
<i>n</i>	8	Segment length used in the fingerprint-based classifier
<i>gap</i>	0	Number of gaps allowed in the fingerprint-based classifier

Table 1.4: Parameters and their base values used in classification experiments.

NumTest is the total number of test sequences, 1,227 in our case. (A test sequence S in a class \mathcal{C} is said to be classified correctly by an algorithm if S is determined to belong to \mathcal{C} by that algorithm.)

Our experimental results showed that for the pattern-based classifier, $B_{high} = 1100$, $B_{low} = 1000$ and $PR = 99.5\%$ and $NR = 0.0\%$. For the fingerprint-based classifier, $B_{high} = 68$, $B_{low} = 65$ and $PR = 99.2\%$ and $NR = 0.8\%$. For comparison purposes, we also ran the FASTA classifier [Lipman and Pearson, 1985; Pearson and Lipman, 1988] on the same datasets. This classifier aligns a given unlabeled DNA sequence S with an ALU consensus taken from the database maintained in the Whitehead Institute for Biomedical Research. It classifies S as an Alu if the alignment score is greater than or equal to a preset threshold. (The score is the best score of S against both strands of the consensus and the threshold used is 100.) Otherwise, S is classified as a non-Alu sequence. The tool does not give the “no opinion” verdict. The experimental results showed that FASTA achieved a 98.2% PR , lower than the PR of our classifiers.

We next conducted a series of experiments to examine the impact of the

parameter values on the performance of the two proposed classifiers. To avoid the mutual influence of parameters, in each experiment we only varied one parameter's values, with the other parameters being fixed and having the values as shown in Table 1.4. It was found that both of the classifiers behave stably with varying $|\mathcal{B}|$, $|\mathcal{T}_P|$ and $|\mathcal{T}_N|$. The relative sizes of the base, positive training and negative training datasets have little impact on the performance of the classifiers, provided that these sets are sufficiently large (e.g., with size ≥ 100). Figure 1.12 shows that the performance of the pattern-based classifier degrades as *Occur* becomes large. We found that the discovered patterns in Alu sequences generally have low occurrence numbers (e.g., with *Occur* ≤ 30). When *Occur* > 30 , very few patterns were discovered and thus they can not well characterize the sequences. Likewise, using short active patterns (e.g., with *Length* = 5) and large distance values (e.g., *Dist* = 3) yields poor performance, since these segments may appear, by chance, in both Alu and non-Alu sequences. No trend is evident with regard to *n* and *gap* used in the fingerprint-based classifier. However, programs using a small *gap* (e.g., *gap* = 0) run much faster than programs using a large *gap* (e.g., *gap* = 4).

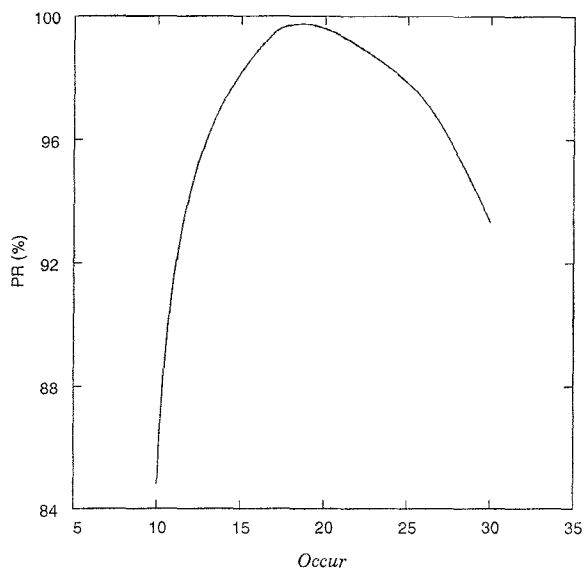
In sum, to achieve good performance, one should use sufficiently many (e.g., 100) sequences as base, positive training and negative training data when constructing the classifiers. For the pattern-based classifier, using long active patterns (e.g., with *Length* = 11) with a small distance value (e.g., *Dist* ≤ 1) is good. Such patterns appear quite uniquely in Alu sequences and therefore characterize the sequences well. For the fingerprint-based classifier, using segments of length 8 without gaps is good, as it requires less running time and space while achieving an acceptably high precision rate.

1.4 Generalizations and Future Work

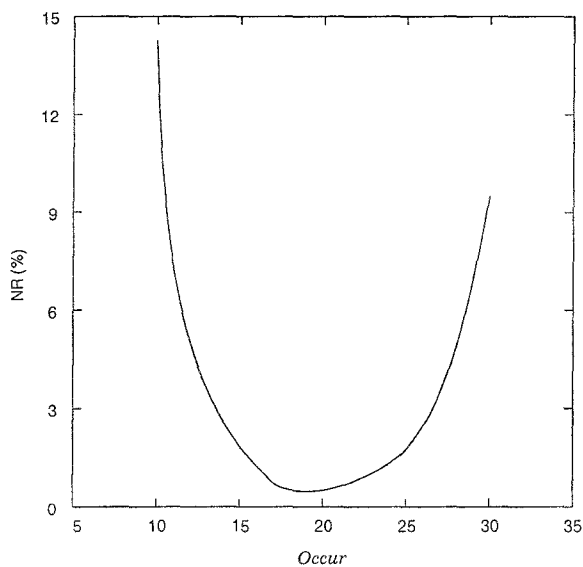
In this chapter we have presented some techniques for pattern discovery and classification in biosequences. Experimental results on both protein and synthetic data showed that our discovery algorithm and optimization heuristics work effectively. Our empirical study also showed a promising result when applying the proposed classification algorithms to the Alu sequences.

Our current work has three main goals.

1. By incorporating a consensus approach into the algorithms presented here, we are developing new techniques to classify more subtle se-



(a)



(b)

Figure 1.12: *PR* and *NR* as a function of *Occur* used in the pattern-based classifier.

quences such as promoters and splice junctions. This work will eventually contribute to gene recognition in DNA.

2. The current classification tools return results without stating the confidence level. We plan to study the statistical behavior of the classifiers so as to enhance the quality of their output.
3. We want to extend our pattern discovery techniques to high dimensional data, such as 3D proteins, and develop new techniques for protein classification and clustering based on the discovered patterns. In this direction we have obtained some preliminary results recently [Wang *et al.*, 1997].

The software developed from the work is available from the authors. Readers interested in obtaining the software can send a written request to jason@village.njit.edu or shasha@cs.nyu.edu or visit our web site at <http://www.cis.njit.edu/~jason/help.html> for details.