# Finding Sequential Patterns From Biological Sequences

A **Major Thesis** submitted

to

**Faculty of Technology**

of

University of Delhi

towards the partial fulfillment of the requirements

for the award of Degree

**Master of Engineering** in

**Computer Technology and Applications**


Submitted by:

**Pradeep kumar**

ME/CTA/2002, University Roll No 4004

Under the Guidance of

**Mrs. RAJNI JINDAL**



Department of Information Technology

Delhi College of Engineering

**Bawana Road, Delhi – 42**

# CERTIFICATE

This is to certify that the major project entitled "**Finding Sequential Patterns From Biological Sequences**" being submitted by **Pradeep Kumar**, **M.E. CTA, Roll. No. 20/CTA/2002, University Roll Number 4004** in partial fulfillment for the award of "Master of Engineering Degree in Computer Technology and Applications" to the Delhi College of Engineering, University of Delhi, Delhi is a record of bona-fide work carried out by him under my guidance and supervision.


**Prof. (Dr.) D. Roy Choudhury**
**Head**
**Department of Computer Engineering**
**Delhi College of Engineering**

**Mrs. RAJNI JINDAL**
**Project Guide**
**Lecturer**
**Department of Information Technology**
**Delhi College of Engineering**

# ACKNOWLEDGEMENT

I am deeply indebted to **Mrs. RAJNI JINDAL**, Lecturer in the Department of Information Technology, Delhi College of Engineering - Delhi, for her valuable guidance and supervision for the completion of the project in hand. Her great expertise and understanding of data mining, algorithms, interest in biological applications, and continued support has been the key for the outcome of current thesis.

I would like to express my sincere thanks for many useful comments and suggestion provided by **Prof. (Dr.) D. Roy Choudhury,** Head, Department of Computer Engineering, Delhi College of Engineering, Delhi.

I wish to express so many thanks and gratitude for all those for their patience and encouragement throughout, who not only inspired me in developing this project but also gave all guidance and suggestions in completing this report.

**Pradeep Kumar**

# ABSTRACT

Bioinformatics became very popular nowadays. Most of the tasks in bioinformatics involve searching of biological databases. The sizes of biological data records are very huge, and the numbers of records in the databases are increasing year by year. So we need efficient searching techniques for biological databases.

Biosequences typically have a small alphabet, a long length, and patterns containing gaps of arbitrary size. Mining frequent patterns in such sequences faces a different type of explosion than in transaction sequences. In this project report, we study how this explosion affects the classic sequential pattern mining, and present a scalable two-phase algorithm to deal with this new explosion.

We propose a new algorithm called Two-Phase Searching Algorithm (2-PSA) that incorporates reliability and efficiency. The first phase "*Segment Phase" first* searches for short patterns containing no gaps, called *segments*. This phase is efficient. The second phase "*Pattern Phase"* searches for long patterns containing multiple segments separated by variable length gaps. This phase is time consuming. The purpose of two phases is to exploit the information obtained from the first phase to speed up the pattern growth and matching and to prune the search space in the second phase. We evaluate this approach on synthetic and real life data sets.

# CONTENTS


Page No.

**1. Introduction**

1.1 Contributions of this work 2

1.2 Thesis Outline 3

**2. Bioinformatics**

2.1What is Bioinformatics used for? 5

2.2 Motivation and background 6

2.3 Problem of Pattern Finding 7

2.3.1 Applications of pattern finding 9

2.3.2 Pattern representation languages 11

2.3.3 Pattern rating functions 13

2.4 Scope of Bioinformatics 15

**3. Definitions**

3.1 Strings 17

3.1.1 Small alphabets 18

3.1.2 Long sequence length 18

3.2 Patterns 18

3.2.1 Deterministic patterns 19

3.2.2 Probabilistic patterns 20

3. 3 Biological motivations for finding patterns 21

3. 4 Finding pattern in proteins. 22

3. 5 Pattern matching 23

**4. Algorithms in Bioinformatics**

4.1 Introduction 24

4.2 Pattern Discovery Algorithms 24

4.2.1 Pattern driven 24

4.2.2 Sequence driven 25

4.3 Sequential Patterns 26

4.3.1 Sequence 26

4.3.2 Support of a pattern 27

4.3.3 Pattern Generation Pruning 28

4.3.4 Pattern Matching Pruning 28

4.3.5 Position query 28

4.4 First Phase Algorithm – SEGMENT 28

4.4.1 Finding base/frequent segments 29

4.4.2 Index-based querying 29

4.4.3 SP-index (Segment-to-Position index) 31
4.4.4 Compression-based querying 31
4.5 Second Phase Algorithm - PATTERN 32
4.5.1 Segment tree (ST) 32
4.5.2 Pattern tree (PT) 33

**5. Proposed Algorithm:** Two-Phase Searching Algorithm (2-PSA)
5.1 Introduction 34
5.2 First-Phase Algorithm 35
5.3 Second–Phase Algorithm 35
5.4 Techniques for comparison 36
5.4.1 Searching Approach 37
5.4.2 Comparison of sequences 38
5.4.3 Characteristics of Sequences 39
5.4.4 Protein family analysis using patterns 41
5.4.5 Regular expressions 41
5.4.6 Expressions for biosequences 42
5.5 Structural sequence patterns 42
5.5.1 Example of structural sequence patterns 43
5.6 Learning – pattern finding 44

**6. Techniques for Biological database searching**
6.1 Introduction 45
6.2 Molecular Biological Background 46
6.3 Biological Databases 51
6.3.1 Sequence Databases
6.3.2 Structural Databases
6.3.3 Collaboration of Databases
6.4 Sequence Matching
6.4.1 Parameters of Sequence
6.4.2 Sequence Matching Methods
6.4.2.1 Dynamic Programming Methods
6.4.2.2 Heuristics Methods
6.4.2.3 Indexing Methods
6.5 Suffix Tree
6.6 Methods for implementation of sequence matching
6.6.1 Sequential computing

6.6.2 Parallel computing
6.6.2.1 Hardware Oriented Approach
6.6.2.2 Software Oriented Approach
6.7.3 Distributed computing

**7. Implementation and experimental results**
7.1 First Phase Algorithm
7.2 Second Phase Algorithm
7.3 Database of DNA

**8. EXTENSION & CONCLUSION**
8.1 Proposed Future Work
8.2 Conclusion

**9. Publicly Available Software Tools**
9.1 Sequence Similarity Search
9.2 Databases of Patterns

**10. References**

# LIST OF FIGURES AND TABLES:


1. Table 1 database of three sequences.
2. Table 2 The Position lists
3. Figure 1 Structure of DNA double helix
4. Figure 2 Synthesis of protein from DNA
5. Figure 3 Three-dimensional structure of a protein
6. Figure 4 A sample data entry of a virus in GenBank
7. Figure 5 A search page from NCBI BLAST website
8. Figure 6 Transformation of the string ACTTAGC
9. Figure 7 Finding the alignment between two strings to gain the maximum score
10. Figure 8 Substitution matrix for nucleotides in DNA sequences
11. Figure 9. A tree on ACATCTTA and suffix tree on the same string
12. Figure 10. Bi-directional data flow in Splash 2 PLA
13. Figure 11. Execution model based on data-parallel execution
14. Figure 12. The overall distributed scheme

# Chapter 1

# Introduction

This chapter provides a short summary of the work presented in the thesis and also some pointers of what each of the following chapters include

This project falls under the broad area of study called *Bioinformatics* or *Computational Molecular Biology* [9,10,11]. Among the various fields of bioinformatics, we are particularly interested in efficient searching and comparison of the biological data. These tasks form a basic part of bioinformatics, and make contributions to many application areas in bioinformatics such as data analysis, data mining, biological reasoning, and evolutional deduction

Bioinformatics is an emerging science, which in its current state can be broadly divided into two categories in much the same way as software: developers and users. Bioinformatics developers range from mathematicians, statisticians, computer scientists, to software engineers and also all of those skills combined - these people are Computational Biologists or 'Bioinformaticians'.

Between these two layers are databases, whose structure is often maintained by developers, but whose content is provided by users. The databases are predominantly government-funded and accessible to the public with a typical browser. Richer institutions often copy the public databases onto a local network computer to speed up access by their local users.

## 1.1 Contributions of this work

We have proposed a new algorithm called Two-Phase Algorithm for finding frequently occurring patterns from sets of sequences.

Biosequence patterns have the form of $X1$ * ……* $X_k$ spanning over a long region, where each $X_i$ is a short region of consecutive items, called a *segment*, and * denotes a variable length gap corresponding to a region not conserved in the evolution [1,2].

The presence of * implies that pattern matching is more permissible and involves the whole range in a sequence. These features create a different type of explosion of patterns. Here we discuss the effect of these features on the classic sequential pattern mining, and propose a two-phase mining strategy to better deal with the new type of explosion.

- The first phase finds frequent segments $X_i$ efficiently.
- The second phase grows patterns $X1$ * …… * $X_{k_i}$ - 1 * $X_k$ rapidly one segment at a time, as opposed to one item at a time. The essence of this two-phase approach is leveraging some information about $Xi$ obtained in the first phase to prune patterns and speed up pattern matching in the second phase.

Particularly, based on such information, we propose indexing or compression methods to reduce the work of pattern matching, and propose a novel pattern enumeration scheme to prune the search space.

## 1.2 Thesis Outline

The thesis is structured as follows. First chapter starts with a brief introduction and outline of this thesis.

Chapter two consists of basics in Bioinformatics and the motivation and background factor of pattern matching of biological sequences. In Chapter 3 we discuss basic definitions and terminology used in bioinformatics. The algorithms in Bioinformatics are described in Chapter 4. The proposed algorithm and Techniques for pattern matching and pattern finding is being presented in chapter five. The techniques for biological database searching have been discussed in Chapter 6.

Implementation of two-phase algorithm and experimental results for finding sequential pattern matching in bioinformatics are described in Chapter 7. Extension and conclusion is described in chapter 8. The Publicly available Software Tools and Databases of Patterns are described in Chapter 9 and finally, the References are presented in Chapter 10.

# Chapter 2

# Bioinformatics

## 2.1 What is Bioinformatics used for?

The Oxford English Dictionary defines bioinformatics as: **"The science of collecting and analyzing complex biological data such as genetic codes"**. Bioinformatics is used for a virtually limitless number of tasks, but some of the most common are [12, 13, 14]:

(1**) Finding homologs ('twins'**) of a gene in your favourite species given a sequence you have in a model species eg. Finding a rice gene given the sequence of an Arabidopsis gene which has been characterized already.

(2) **Comparing the similarity between two or more gene sequences to get a measure of their relatedness.** This can be used to group genes into subsets (orthological, paralogical - ) which might give an indication of the function or activity of the members of these subsets based upon what is already known about the proteins encoded by the membership of that subset. Comparisons also allows taxonomy to be examined, as well as the drawing of **phylogenetic trees (trees of relatedness**) and insights can be made into sequence evolution.

(3) **Design of primers** Online and offline tools allow individuals and whole projects (eg. sequencing projects) to have their computers design thousands of primers with little effort. The primers are then used to sequence or amplify unknown or interesting genes or gene sequence.

(4) **Reconstructing genes from EST sequences**. **E**xpressed **S**equence **T**ags are short pieces of genes which are expressed, which have been cloned and sequenced - then deposited into the public gene databases.

(5) **Grouping of proteins into families**. There are is a huge amount of work being undertaken to classify the proteins encoded by genes into super families and families

## 2.2 Motivation and background

The amount of the data collected and stored in databases worldwide is growing with increasing speed. While computers were initially designed and used mostly for numerical computations, a large proportion of the data is nowadays collected and processed in textual form [7, 12,13]. The sources of textual data can be very different, varying from documents in natural languages to the sequences of biological macromolecules. Efficient methods of analysis are required for understanding the underlying principles about the sequence data.

Perhaps one of the most fascinating languages being studied by the mankind is our own genetic code, the DNA, RNA, and protein molecules that are essential to all life on planet Earth. These macromolecules are built (usually in a linear manner) from a small amount of building blocks like nucleotides or amino acids.

The genetic code, using perhaps an oversimplification, can thus be interpreted as sequences of these basic building blocks or letters. The function of each of the molecules is usually determined by their structure, and one of the key questions in

modern molecular biology is to understand the relationship between the sequence, structure, and function of these molecules as well as the biological processes they are involved in.

The machinery that is able to read, interpret, replicate, and otherwise utilize the information stored in DNA, RNA and protein molecules are essential to life. This universal language of life has evolved over billions of years, producing many different life forms from the simplest bacteria to human beings, being able to survive in extreme heat and pressure or under constant freezing conditions, or consuming completely different energy sources. Many of the properties of our genetic code and the ways to interpret that code remain yet to be discovered, described, and understood.

In this thesis we study pattern discovery approaches for finding regularities in the sequence data. The main focus is on discovering patterns, the words or sentences according to some linguistic rules, that occur frequently in input sequences or are characteristic for certain subsets of the input data. Firstly, the frequently recurring patterns are often indicative of the underlying structure and function, as in biology, the conservation of certain features in the course of evolution usually indicates the importance of these features. Secondly, different subsets of input data may represent examples from meaningful concepts. Patterns common to these different subsets can help to distinguish between these sets, as well as to reveal features important for different classes of sequences.

**These two cases of pattern discovery describe the two basic problems**

- The *family conservation problem* and
- The *family classification problem*, which both are discussed in the thesis.

In this thesis we focus on pattern discovery in biological sequences, as this is perhaps one of the most important application areas with implications to molecular biology and medicine. The aim of the current research is to develop methods for discovering patterns that can be used for advancing the biological knowledge about the structure and function of the genes and gene products.

## 2.3 Problem of Pattern Finding

Sequence pattern matching is a research area aiming at developing tools and methods for finding *a priori* unknown patterns in a given set of sequences, patterns that are frequent, unexpected, or interesting according to some formal criteria.

Patterns are formal grammatical descriptions for certain *languages* representing subsets of all possible sequences over a finite alphabet. Patterns can be represented using different formalisms, for example, as regular expressions, or probabilistic weight matrices [12, 16]. The *interestingness* of patterns can be interpreted in relation to the pattern description itself or in relation to the sequences being analyzed. For example, if the pattern occurs significantly more frequently than expected by chance then the pattern may be considered interesting. In order to define the interestingness of a pattern a formal scoring mechanism is needed. And finally, for practical applications the algorithms and tools are needed that can be used for discovering interesting patterns.

Overall, the **pattern finding problem can thus be divided into three sub problems:**

1. Choosing the appropriate language to describe patterns

2. Choosing the scoring function for comparing patterns

## 3. Designing an efficient algorithm for identifying the best-scoring patterns

Appropriate language for describing patterns depends from the application area. For example, one can ask if the type of patterns one is looking to discover can in principle capture the biological phenomena one attempts to study. Thus, the pattern language has to be chosen appropriately from the biological point of view. Similarly, the scoring function has to be such that it has proven relevance also in the biological terms, not only in abstract mathematical or statistical sense. Unfortunately, not all pattern languages and scoring functions are such that efficient algorithms can be designed so that the pattern discovery could be performed in a reasonable time.

From the practical point of view, one may have to balance between the desire to use the very complex pattern language and scoring functions, which are biologically perhaps the most relevant, and the available compute resources that require to use computationally more feasible methods. For very large data sets, for example, one may have to use simpler pattern representation language that can offer improved speed in calculations. The balance has to be reasonable though, so that the chosen pattern language and scoring functions do not eliminate the biological relevance of the discovered patterns.

In current thesis we have developed methods and tools for the exhaustive search for the best patterns from a range of different pattern representation languages. In the practical applications we also demonstrate that the choice of simple pattern languages is often sufficient to capture rather complex biological information. This is the motivation throughout the thesis, to design practical algorithms that can be used for studying biologically relevant pattern classes in a variety of biological applications.

Before discussing the three aspects of pattern discovery in more detail we present a few practical applications of using the patterns to represent biologically meaningful concepts.

### 2.3.1 Applications of pattern finding

Biological sequences, or *biosequences*, can be grouped in *families* based on their function, structure, cellular location, molecular processes, gene regulation, or other criteria [1, 16]. Here we present some applications, where the patterns common to these groups are able to capture very different biological features.

#### Sequence analysis

Many of the protein families and their characteristic patterns have been collected in the protein family database PROSITE. Finding characterizations of biosequence families is an important sequence analysis problem. If a feature common to all known sequences of a family is found, then it is likely that this particular feature is important for the biological role of the family. Algorithms for sequence pattern discovery have been widely used for characterizing protein families.

#### Structural information

Jonassen and colleagues have studied patterns that incorporate structural information about the packing of *residues*, *i.e.* amino acids in the protein sequence, in three-dimensional space. They define a *packing motif* as a pattern that has multiple occurrences in a set of protein structures. Packing motifs describe clusters of residues that are spatially close together in the 3-D structure, but not necessarily in the primary sequence.

#### Finding Patterns

Patterns in protein sequences can represent potentially important features for their functional activities. We have applied pattern discovery combined with careful targeted input sequence selection for predicting the coupling specificity of specific transmembrane receptor proteins called G-protein coupled receptors (GPCR) and the G-proteins from $G_s$ , $G_{i/o}$ =o , or $Gq_{11}$ class.

The task of pattern discovery is to predict the potential regulatory signals, for example the transcription factor binding sites, from the DNA. From the computer science viewpoint considering pattern discovery as pure string algorithms, the DNA and proteins differ only by the alphabet size (four and twenty, respectively). Yet, these sequences do represent different physical objects and hence the need for finding patterns may arise in different biological research domains.

**Research & development**

Often the respective research communities are separated, as well as the approaches developed. The biological features represented by patterns can vary in semantics depending on the biological application, and hence the language of representing the patterns and the criteria for evaluating their interestingness can be very different for different applications.

Usually, the data sets involving DNA sequences are much larger than those for the proteins. Finally, the physical-chemical properties of the real atoms represented by letters of an alphabet, or other physical constraints of the molecules in the different application domains differ and may need to be taken into account in pattern discovery.

### 2.3.2 Pattern representation languages

According to the pattern language we can distinguish between discrete patterns like regular expression type motifs and probabilistic patterns like probabilistic weight

matrices. Here we discuss the deterministic regular patterns and approximately matching patterns[12, 14].

Although the probabilistic motif representation is more appropriate for describing certain physical features of the molecules, like a protein's binding efficiency to DNA, these motifs are more complex to discover by computational methods due to a much larger search space.

One of the oldest and most prominent pattern databases, the PROSITE database stores information about protein families, their descriptions, and patterns that can be used to determine the membership of novel sequences to these families. Biologically significant patterns and profiles are formulated in such a way that with appropriate computational tools they can help to determine to which known family of proteins the new sequence may belong, or which known domain(s) it contains.

In this section we provide as an example the definition of the pattern language as used in the PROSITE database, as well as give two examples of the PROSITE entries showing how the patterns from this pattern language can capture biologically relevant features about real protein families.

**Example 2.1** *Pattern definitions from the PROSITE database* ***(http://www.expasy.org/prosite/).***

The PA (PAttern) lines contain the definition of a PROSITE pattern. The patterns are described using the following conventions:

- The standard IUPAC one-letter codes for the amino acids are used.
- The symbol 'x' is used for a position where any amino acid is accepted.

- Ambiguities are indicated by listing the acceptable amino acids for a given position, between square parentheses '[ ]'. For example: [ALT] stands for Ala or Leu or Thr.

- Ambiguities are also indicated by listing between a pair of curly brackets '{ }' the amino acids that are not accepted at a given position. For example: {AM} stands for any amino acid except Ala and Met.

- Each element in a pattern is separated from its neighbor by a '-'.

- Repetition of an element of the pattern can be indicated by following that element with a numerical value or a numerical range between parentheses. Examples: x(3) corresponds to x-x-x, x(2,4) corresponds to x-x or x-x-x or x-x-x-x.

- When a pattern is restricted to either the N- or C-terminal of a sequence, that pattern either starts with a '<' symbol or respectively ends with a '>' symbol.

- A period ends the pattern.

  Examples: PA : [AC] _ x _ V _ x(4) _ {ED}:

This pattern is translated as: [Ala or Cys]-any-Val-any-any-any-any-{any but Glu or Asp }

PA : < A _ x _ [ST](2) _ x(0; 1) _ V:

This pattern, which must be in the N-terminal of the sequence ('<'), is translated as: Ala-any-[Ser or Thr]-[Ser or Thr]-(any or none)-Val.Using this syntax for possible patterns in protein sequences, the sequence families can be described.

Similar types of patterns can also be used for analyzing DNA sequences. The DNA-binding proteins are known to bind to specific parts of DNA, which can be described in terms of sequence motifs. For example, the pattern GGTGGCAA which has been shown to be a protease specific control element.

### 2.3.3 Pattern rating functions

Given a family of related sequences, there may exist many patterns that are present in all or nearly all of the sequences [14,16]. The more complex the pattern language, the more different patterns match at least some of the sequences. It is a challenging task to tell which of these patterns are relevant. For sorting the patterns according to their interestingness and relevance we need formal *fitness measures* that give to each pattern a score that can be used for comparing patterns.

These fitness measures can be based [10] on the specificity and sensitivity of the patterns, the information content, the ratio, the probability statistics, the minimum description length (MDL) principle, and others. Sometimes several simple quality indicators can be presented to users, as in the following example from PROSITE.

The aim of different pattern discovery methods is usually to find motifs that are overrepresented in the data set analyzed, or unexpected according to some other criteria. It is possible to count how many sequences contain the motif or how many occurrences of the motif there is in total (*i.e.* count numbers of occurrences within the same sequence). When counting several occurrences within each sequence the occurrences may be overlapping and not independent. Therefore, it is simpler to count just the number of sequences that contain the motif.

The ratio of pattern occurrences in two data sets tells how much more frequent the pattern is in one data set than in another. The problem with ratios is that if the

frequencies are small then the ratios may be very high, even though the patterns do not represent meaningful concepts. These high ratios may be slightly compensated by assuming higher expected number of occurrences in the comparison set.

Given the background model for the expected number of occurrences, for example from the explicit counting of pattern occurrences in comparison data, one can estimate how many occurrences of each pattern to expect.

This estimate can be used to calculate how probable the actual number of occurrences is (assuming the same background model) based on binomial or hyper-geometric distribution, for example. Binomial distribution assumes independent random trials and allows to calculate the probability to observe each pattern at least a given number of times in the data.

Hyper-geometric distribution corresponds to selection without replacement, *i.e.* the probabilities depend on previous outcomes. For large data sizes and small numbers of trials binomial distribution approximates well the hypergeometric distribution. When calculating the total number of occurrences for patterns, *i.e.* possibly several occurrences per one sequence, one can in principle use the same statistical criteria.

However, one has to be aware of the possibility that pattern occurrences may be overlapping and thus not independent. The cyclic patterns, *i.e.* patterns that can have an overlap with themselves, have a higher expected number of occurrences even under the assumption that all nucleotides have equal and independent probability of occurrence at each position.

## 2.4 Scope of Bioinformatics

Bioinformatics means solving problems arising from biology using methods from computer science. The goal is to understand the functioning of living things, and to

improve the quality of life [9, 10, 11]. This field has become very popular since 1980s.

**There are many sub-areas in bioinformatics:**

- Data comparison
- Data analysis
- DNA assembly
- DNA mapping
- Gene finding
- Evolutional deduction
- Protein structure prediction
- Data visualization
- Data mining
- Drug design
- Statistical genetics etc.

All these areas are more or less related. However, among these, the area of data comparison is most relevant to this project report.

The biological data (sequences and structures) are naturally very huge. For example, a DNA sequence record includes 50 to 250 million characters. In addition, the numbers of records in the biological databases are dramatically increasing year by year because of the intensive researches in the field of molecular biology. So, it is totally impossible to search through these data without the help of computers. Indeed, their sizes are so enormous that it is even impossible for naive computer algorithms to carry out this job. So, we need “smart” algorithms to handle these large data efficiently.

Database searching task mainly involves comparison of biological data. Sequence comparison is needed in the case of sequence data, and structural (3D) comparison in the case of structural data. The perfect solutions for these comparison tasks have shown to be very high in time complexities. But, in biology, the approximate results are also quite useful. So, algorithms that provide sub-optimal solutions (i.e. with certain percentage of errors) within a reasonable time can still be extremely useful in many practical problems.

# Chapter 3

# Definitions

Pattern discovery deals with methods for finding regularities in sequences. Here we define the concepts of sequences, patterns and provide the basic framework used later for the design of algorithms for pattern discovery.

### 3.1 Strings

We use Σ to denote a finite set of characters, an *alphabet*. The *size* of the alphabet Σ is | Σ |. Any sequence $S = a_1a_2a_3 \ldots\ldots\ldots a_n$ such that n >= 0 and each $a_i$ is in Σ is called a *string* (or *sequence*, or *word*) over the character set Σ. The *length* |S| of the string S is n. The string of length 0, *i.e.* an empty string, is denoted by λ. The set of all possible strings over Σ is $\Sigma^k$ [5,6].

We identify individual characters by their positions within the string. The character $a_i$ at the position i can also be denoted by S[i]. Character positions of a non-empty string

S are in the range 1 <= i < = |S| , *i.e.* the first character of the string is at position 1, and the last character is at position |S|.

Consecutive characters $a_i$ ……..$a_j$ of S form a *substring* of S that starts from position i and ends at position j. We denote this substring by S[i..j], where 1<= i <=j<=|S| . An alternative definition which does not use character positions within the string states that x is a substring of S if S = yxz for some strings y and z.

A substring S[i..i] has length 1 and corresponds to the character $a_i$ at position i. A substring S[i..j] has length j - i + 1. We say that substring S[i..j] *occurs* at the *position* or *location* j of the string S. We say that a substring x has multiple occurrences in S if x = S[i..j] = S[i'..j'], and j' != j'.

**3.1.1 Small alphabets**: Biosequences have a very small alphabet, i.e., 4 [ A T C G ] for DNA sequences and 20 for protein sequences, and many short patterns occur in most sequences. In contrast, transaction sequences have a large alphabet, ranging from 1,000 to 10,000, and only a tiny fraction of items occurs in a transaction sequence. With most items occurring in every biosequence, pruning strategies and data structures based on the sparsity or absence of items, such as the hash-tree and the idlist or bitmap representation are not effective for biosequences.

**3.1.2 Long sequence length**: A biosequence has a typical length of few hundreds, sometime thousands. In contrast, a transaction sequence has a typical length from 10 to 20. A long sequence (especially, with a small alphabet) often contains long patterns. The classic sequential pattern growth of one item at a time requires many database scans and high frequency of pattern matching.

## 3.2 Patterns

Pattern finding is one of the fundamental problems in bioinformatics. It can be used in multiple sequence alignment, protein structure and function prediction, characterization of protein families, promoter signal detection, and other areas.

One important problem arising from bio-applications is the discovery of sequential patterns that occur in many biosequences in a given database (i.e., DNA or protein sequences). Such frequent patterns typically correspond to residues conserved during evolution due to an important structural or functional role.

Finding frequent patterns often is the first step in sequence analysis such as classifying sequences, extracting species-specific features, identifying transcription factor binding sites, etc. We focus on the scalable techniques for mining frequent patterns from a large database of biosequences.

In biology, various tools have been developed for searching for similarity among biosequences. A well known tool is BLAST (Basic Local Alignment Search Tool). The idea is *aligning* sequences so that similarity can be revealed in the presence of small variations in position.

Sequential pattern mining developed in data mining searches for all frequent patterns in transaction sequences" motivated in marketplaces. A transaction sequence can be a purchase sequence, a web link click stream, etc. The focus of those works is on the scalability on large databases. A natural solution is to sequential pattern mining to biosequences.

**Types of patterns**

Different programs discover patterns of different kind. On the most general level patterns can be divided between deterministic and probabilistic. A deterministic pattern either matches given string or not. On the other hand probabilistic patterns are

usually probabilistic models that give to each sequence probability that this sequence is generated by the model. The higher is this probability; the better is the match between sequence and pattern [7,8].

### 3.2.1 Deterministic patterns

The simplest kind of a pattern is just a sequence of characters from alphabet Σ , such TATAAAA, the TATA box consensus sequence. We can also allow more complex patterns, adding some of the following frequently used features.

- **Ambiguous character** - is a character corresponding to a subset of Σ . Ambiguous character then matches any character from this set. Such sets are usually denoted by a list of its members enclosed in square brackets e.g. [LF] is a set containing L and F. A-[LF]-G is a pattern in a notation used in PROSITE database. This patterns matches 3-character subsequences starting with A, ending with G and having either L or F in the middle.

  For nucleotide sequence there is a special letter for each set of nucleotides, where R=[AG], Y=[CT], W=[AT], S=[GC], B=[CGT], D=[AGT], H=[ACT], V=[ACG], N=[ACGT].

- **Wild-card or don't care -** is a special kind of ambiguous character that matches any character from Σ . Wild-cards are denoted by N in nucleotide sequences, X in protein sequences. Often they are also denoted by dot '.'. Sequence of one or several consecutive wild-cards is called gap and patterns allowing wild-cards are often called gapped patterns [1].

- **Flexible gap** - is a gap of variable length. In PROSITE database it is denoted by x(i,j) where i is the lower bound on the gap length and j is an upper bound. Thus x(4,6) matches any gap with length 4, 5, or 6. They also denote a fixed

gap of length i as x(i) (e.g.(3) = …...). Finally * denotes gap of any length (possibly 0).

- **Patterns with mismatches** - One can further extend expressive power of deterministic patterns by allowing certain number of mismatches. Most commonly used type of mismatches are substitutions. In this case subsequence S matches pattern P with at most k mismatches, if there is a sequence S0 exactly matching S that differs from S in at most k positions.

### 3.2.2 Probabilistic patterns

The simplest type of probabilistic pattern is position-weight matrix (PWM). PWMs are also sometimes called position-specific score matrix (PSSM), or a profile (however profiles are often more complicated patterns, allowing gaps). PWM is a simple ungapped pattern specified by a table. This table contains for each pair (position; character), the relative frequency of the character at that position of the pattern.

Assume that the pattern (i.e. PWM) has lent k. The score of a sequence segment x1 …….. $x_k$ of length k is

$$\Pi = \prod_{i=1}^{k} A[xi; i] / f(x_i)$$

where A[c; i] is an entry of position weight matrix corresponding to position i of the pattern and character c and f(c) is background frequency of character c in all considered sequences. This product represents odd-score that the sequence segment x1 … $x_k$ belongs to the probability distribution represented by the PWM.

### 3. 3 Biological motivations for finding patterns

Nucleotide and protein sequences contain patterns that have been preserved through evolution because they are important to the structure or function of the molecule [10].

In proteins, these conserved sequences may be involved in the binding of the protein to its substrate or to another protein, may comprise the active site of an enzyme or may determine the three dimensional structure of the protein.

Nucleotide sequences outside of coding regions in general tend to be less conserved among organisms, except where they are important for function, that is, where they are involved in the regulation of gene expression. Discovery of motifs in protein and nucleotide sequences can lead to determination of function and to elucidation of evolutionary relationships among sequences [11].

### 3. 4 Finding pattern in proteins

With the accumulation of nucleotide sequences for the entire genomes of many different organisms, comes the need to make sense out of all of the information. Attempts have been made to organize all of the proteins encoded in these genomic sequences into families based on the presence of common signature sequences.

Members of protein families are often characterized by more than one motif (on average each family has 3-4 conserved regions) which increases the certainty that a protein has been assigned to a correct family. Hierarchical trees of protein clusters often reveal functional and evolutionary relationships among proteins. Starting with a single "seed" sequence, protein families can be characterized in order to find ancient ancestor sequences [11,13].

First, proteins related to a query sequence are found by searching the databases for similar sequences. Sequences revealed from this initial screen are then used as query sequences to search for other family members and the process is repeated to exhaustion.

All of the sequences are aligned in order to identify conserved regions which are used to generate models that represent ancient conserved regions. The rationale behind this approach is that if protein A is related to protein B, and B is related to C, then A is also related to C.

By this method, proteins are assigned to a family based on sequence homology as determined primarily by alignment. If an alignment finds homology between a query protein and a particular family of proteins, a phylogenetic relationship between them is automatically assumed. There are two problems with this assumption:

1) Significant sequence similarities are not always indicative of close evolutionary relations.

2) Despite limited sequence homology, proteins can have structural and mechanistic similarities, and even common ancestry not apparent through alignment. Perhaps structural information should also be considered when attempting to classify proteins that are highly divergent in homology, yet functionally equivalent.

### 3. 5 Pattern matching

The problem of pattern discovery, i.e. the algorithm is supposed to discover pattern unknown in advance. However in biology many consensus sequences are known and it is important to have tools that allow to find occurrences of known patterns in new sequences.

This problem is called pattern matching. Program for pattern matching can be quite general, i.e. they get pattern as a part of input, or they can be built to recognize only one particular kind of pattern.

# Chapter 4

# Algorithms in Bioinformatics

## 4.1 Introduction

The following are some of the most important algorithmic trends in Bioinformatics [6, 10] :

1. Finding similarities among strings (such as proteins of different organisms)
2. Detecting certain patterns within strings (such as genes, introns, and □-helices)
3. Finding similarities among parts of spatial structures (such as motifs)
4. Constructing trees (called phylogenetic trees) expressing the evolution of organisms whose DNA or proteins are currently known

5. Classifying new data according to previously clustered sets of annotated data

6. Reasoning about micro array data and the corresponding behavior of pathways.

## 4.2 Pattern discovery algorithms

Pattern discovery algorithms can be divided into two groups:

- Pattern driven and
- Sequence driven.

### 4.2.1 Pattern-driven (PD)

Pattern-driven approaches are based on enumerating candidate patterns and selecting those with the best fitness; the general framework of these algorithms is:

(1) Define the solution space, i.e. a set of patterns, and the fitness measure

(2) Enumerate the patterns in the solution space

(3) Calculate the fitness of each pattern with respect to the given examples

(4) Report the fittest patterns

The most straightforward implementation is to limit the solution space by the size of the patterns, and to explicitly enumerate all the patterns from this space one by one. The advantage of this approach is that it is possible to guarantee finding the best patterns up to some limited size, almost regardless of the total length of the examples. This is because it is usually possible to organise the algorithm so that it is linear-time in this length.

On the other hand the size of the pattern-space is exponential in the length of the patterns - for example there are more than 1013 different sub-string patterns of length

10 over the amino acid alphabet. PD algorithms guaranteed to find the pattern with the highest fitness value, have worst case time complexity exponential in the length of the patterns.

### 4.2.2 Sequence or structure-driven (SD)

This method finds patterns by comparing given strings or structures and then looking for local similarities between them. For instance an SD algorithm may be based on constructing a local multiple alignment of given sequences and then extracting the patterns from the alignment by combining the segments common to most of the sequences [16]. This may be achieved by

(1) Grouping the sequences according to sequence similarity

(2) Finding a common pattern, e.g. by dynamic programming that matches all or most of the sequences described by the parent groups

(3) Grouping similar patterns together and repeating step (2) until only one group is left.

In general, more than two patterns may be combined in step (3). SD methods also differ in how the sets to be combined are chosen, how combination is performed (dynamic programming, heuristics) and how the (fittest) patterns are chosen, how patterns are represented and how many patterns are kept from stage (2).

It may be possible to discover patterns of an almost arbitrary size by SD algorithms. However, since the construction of an optimal alignment or finding the longest sub-sequence are NP-hard problems, SD methods have to be based on heuristics and hence cannot guarantee optimal results. In general SD algorithms tend to work well if the sequences are sufficiently similar.

## 4.3 Sequential Patterns

Pattern finding programs usually consists of several sequences, some basic terminology used in this project are discussed below.

### 4.3.1 Sequence

**Input sequences**

The input of pattern finding programs usually consists of several sequences, expected to contain the pattern. We denote Σ the alphabet of all possible characters occurring in the sequences [6, 10, 14]. Thus Σ = {A, C, G, T } for DNA sequences and Σ is a set of all 20 amino acids for protein sequences. Most of the algorithms can be easily adapted to work with any finite alphabet (this is true for algorithms, but not necessarily for their implementations). Thus the pattern finding algorithm can be used also outside bioinformatics, or on other types of biological data.

A database D is a collection of sequences {s1, …….sN }. Each sequence **si** is an ordered list of items chosen from a fixed alphabet. < si; j > denotes the jth position in a sequence si, where j >= 1.

A *segment* refers to one or more items at consecutive positions in a sequence. A *pattern* has the form *X*1 * ……. *Xn* (*n* >= 1), where *Xi* is a segment and * denotes the variable length “don't care" (VLDC). A pattern *X*1* ……*Xn matches* a sequence *si* if each segment *Xj* matches itself and each * can substitute for zero or more items.

Useful patterns for sequences in *D* should occur frequently in sequences in *D*, but not in other sequences. For long sequences over a small alphabet, a segment *Xi* of a short length tends to occur in every sequence, similar to “stop words" that occur in every

text document. Such trivial similarity is not discriminating, therefore, not useful for biology analysis. For example, it is known to biologists that a transcription factor binding site has a length from 6 to 15 . We can specify a minimum segment length to exclude trivial segments.

### 4.3.2 Support of a pattern

Support *of a pattern is the percentage of the sequences in D that* contain the pattern [1, 16]. Given a minimum segment length MinLen and a minimum support MinSup, *a pattern* *X*1 * …… *Xn is* frequent *if* | *Xi* | >= *MinLen for 1* <= *i* <= *n* ¸ *MinLen for and the support of the pattern is at least MinSup. The problem of* mining sequence patterns *is to find all frequent patterns.*

We find all frequent patterns in two phases. The first phase, *Segment Phase*, finds all frequent segments *Xi* satisfying the minimum length. The second phase, *Pattern Phase*, generates frequent patterns *X*1 * ……..* *Xk* using *Xi found* in the first phase.

### 4.3.3 Pattern Generation Pruning:

If *P* * *X* fails to be a frequent pattern, so does *P'* * *X*. Therefore, we can prune *P'* * *X*.

### 4.3.4 Pattern Matching Pruning

If *P** *X* fails to occur before position *I* in sequence *s*, so does *P0* ¤ *X*. Therefore, we only need to examine the positions after *i* when matching *P0* ¤ *X* against *s*. To support these prunings, we need a strategy for enumerating the pattern space *X*1 ¤¢ ¢ ¢¤*Xk* so that *P* is enumerated before *P0*, and we need to answer the following queries efficiently [1].

### 4.3.5 Position query

Q(X; s; i): given a frequent segment X, a sequence id s, and a position i in s, find the smallest start positio n of X in s greater than i. If such a position j is found, return < s; j >; otherwise, return nil [1].

### 4.4 First Phase Algorithm - SEGMENT PHASE

This phase finds all frequent segments and builds an auxiliary structure for answering position queries.

#### 4.4.1 Finding base/frequent segments:

We use the *generalized suffix tree (GST)* to count support of segments. The time and space needed for constructing the GST is $O(|D|)$, where $|D|$ *is* the total length of the sequences in $D$. We extract the following information from the GST.

(1) The frequent segments of length *MinLen*, *Bi*, called *base segments*, and the *position lists* for each *Bi*,

$s : p1 , p2 , \ldots\ldots,$ where $pj < pj+1$ and each $< s , pj >$ is a start position of *Bi*.

(2) All frequent segments of length *greater than MinLen [1]*.

(We do not extract position lists for such frequent segments.)

#### 4.4.2 Index-based querying

In this method, we build an in-memory index for the positions [1, 6,10] of base segments. First, we rewrite each frequent segment $X$ using base segments only. Consider two base segments $B1$ and $B2$, such that the last $k$ items in $B1$ are identical to the first $k$ items in $B2$ where $k >= 0$. The *k-join* of $B1$ and $B2$, denoted by B1 |×|k B2 , is the segment obtained by overlapping the last $k$ items of $B1$ with the first $k$ items of $B2$.

| ID | SEQUENCE |
|---|---|
| $s_1$ | **abacdab** |
| $s_2$ | **abcacda** |
| $s_3$ | **baacdca** |

**Table 1: The database D**

**Example 4.1**

Table 1 shows a database of three sequences, with the alphabet { a , b , c , d }.

Let MinSup = 2/3, and MinLen = 2.

The following segments are frequent:

ab(2), ac(3), acd(3), acda(2), cd(3), cda(2), da(2).

The integers in the brackets are support counts. The base segments and their position lists are given in Table 2. ab * cda occurs in s1 and s2, so is a frequent pattern. We can write ab * cda as B1 * (B3 11 B4) using only base segments. Similarly, ab * acda is frequent and can be written as B1 * (B2 10 B4).

| Base Segments | Position Lists |
|---|---|
| **B1 = ab** | **($s_1$ : 1,6) , ($s_2$ :1)** |
| **B2 = ac** | **($s_1$ : 3) , ($s_2$ : 4 ), ($s_3$ : 3)** |
| **B3 = cd** | **($s_1$ : 4) , ($s_2$ : 5 ), ($s_3$ : 4)** |
| **B4 = da** | **($s_1$ : 5) , ($s_2$ : 6 )** |

**Table 2: The Position lists**

We build the following index using the position lists of base segments.

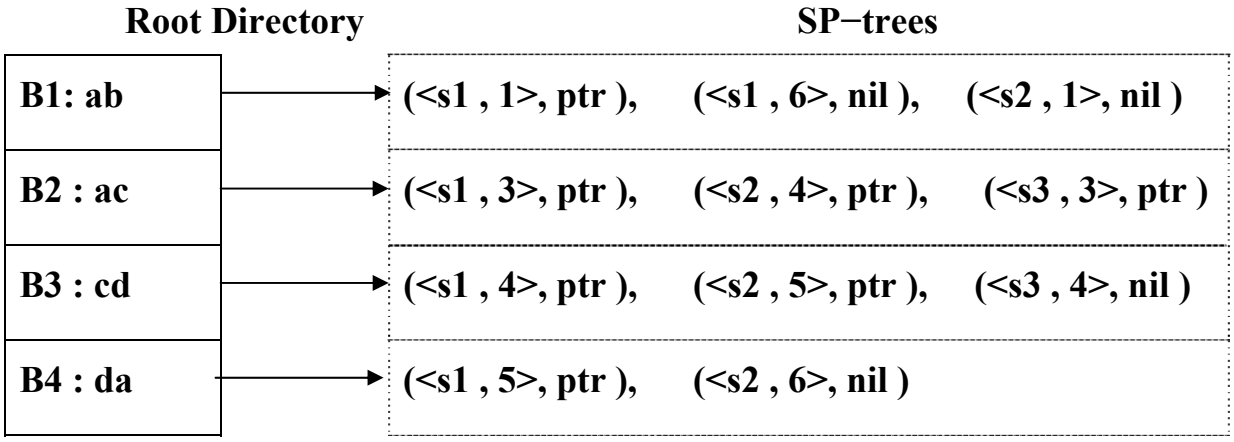



**Figure 1: The SP-index in Example 4.1**

### 4.4.3 SP-index (Segment-to-Position index)

SP-index (Segment-to-Position index) has two components [1]

- The root directory and
- The SP-trees.
- For each Bi, the root directory has an entry for the root of the SP-tree for Bi.
- The SP-tree for Bi is a B-tree for indexing the start positions < s, p > of Bi in all sequences s.
- A leaf entry has the form (< s, p >, ptr). Unlike the standard B-tree, ptr points to the leaf entry (< s, p' >, ptr') for the next base segment in Corollary 1 if there is one, or else nil.

### 4.4.4 Compression-based querying

This method compresses all positions in a non-coding region into a new item ε that matches no existing item except *. A non-coding region contains no part of a frequent segment[1].

We can scan each original sequence once, identify each consecutive region not overlapping with any frequent segment, collapse it into the new item ε. For a long sequence and large MinLen and MinSup, a compressed sequence is typically much shorter than the original sequence. To answer the query Q(X, S, i) over a compressed sequence S, we scan S sequentially because S is short. Note that ε in S does not match any item in X[1].

Example 4.2. For the database in Example 4.1, the compressed sequences for s1; s2; s3 are:

**S1 : abacdab.**

**S2 : ab ε acda (c collapses into ε).**

**S3 : acd (ba and ca collapse into leading ε and ending ε ,** which are deleted)

The compression-based querying is amenable to approximate pattern matching [1].

## 4.5 Second Phase Algorithm - PATTERN PHASE

This phase generates all frequent patterns $X_1$ * ……..* $X_k$ using frequent segments Xi found in Segment Phase. The key is to organize the search space for patterns $X_1$ * ……..* $X_k$ so that the Pattern Generation Pruning and Pattern Matching Pruning mentioned can be easily exploited. The segment tree and pattern tree defined below describe this organization [1]

### 4.5.1 Segment tree (ST)

The ST organizes frequent segments X into a tree so that if X is a prefix of X' then X is enumerated before X' in the depth-first enumeration of the tree. A terminal edge is labeled by an integer k >= 0.

A non-root node w is labeled by a base segment Bi, and represents the frequent segment $B_1 \ |X|_0 \ldots\ldots\ldots\ldots |X|_0 \ B_{p-1} \ |X|_k \ B_p$, where $B_1$ , …….$B_{p-1}$, k , $B_p$ are the labels on the path from the root to w. Let seg (w) denote the frequent segment represented by w[1].

Example 5.1. Figure 2 shows the ST for Example 4.1, with $w_i$ denoting the ith node in the depth-first enumeration of the ST. $w_3$ represents the frequent segment seg($w_3$) = $B_2 \ |X|_1 \ B_3$ = acd, where $B_2$ , 1, $B_3$ are the labels on the path from the root to $w_3$.

$w_4$ represents the frequent segment seg($w_4$ ) = $B_2 \ |X|_0 \ B_4$ = acda, and $w_6$ represents the frequent segment seg($w_6$) = $B_3 \ |X|_1 \ B_4$ = cda..

### 4.5.2 Pattern tree (PT)

The PT organizes patterns $X_1$ * ……..* $X_k$ into a tree so that a super-pattern is enumerated after a sub-pattern in the depth-first [1,6] enumeration of the tree. A non-root node v is labeled by a frequent segment seg($w_i$), where $w_i$ is a node in ST, and represents the pattern seg($w_1$)* ……* seg($w_k$ ), where seg($w_1$), …….. seg($w_k$) are the labels on the path from the root to v. Let pat(v) denote the pattern represented by v.

Furthermore, if $v_1$ , …….. $v_n$ are the child nodes from left to right, with the labels seg($w'_1$ ), …… seg ($w'_n$ ), where $w'_1$, ……$w'_n$ , are in the order of depth-first enumeration of ST.

Therefore, if (non-root node) w is the parent of w' in ST (therefore, seg(w) is a prefix of seg(w0)), the node for P = $X_1$ * ……..* $X_{k-1}$ * seg(w) is the immediate left sibling of the node for P' = $X_1$ * …….* $X_{k-1}$ * seg(w') in PT, therefore, P is enumerated before P' in the depth-first enumeration of PT [1].

Below, we sketch our algorithm of using this property to perform Pattern Generation Pruning and Pattern Matching Pruning.

# Chapter 5

# Proposed Algorithm: Two-Phase Searching Algorithm (2-PSA)

## 5.1 Introduction

In the problem of pattern matching the algorithm is supposed to discover pattern unknown in advance. However in biology many consensus sequences are known and it is important to have tools that allow to find occurrences of known patterns in new sequences. Programs for pattern matching can be quite general, i.e. they get pattern as a part of input, or they can be built to recognize only one particular kind of pattern

We propose a new algorithm called **Two-Phase Searching Algorithm (2-PSA)** that incorporates reliability and efficiency. The first phase "Segment Phase" searches for short patterns containing no gaps, called *segments* . This phase is efficient. The second phase "*Pattern Phase"* searches for long patterns containing multiple segments

separated by variable length gaps. This phase is time consuming. The purpose of two phases is to exploit the information obtained from the first phase to speed up the pattern growth and matching and to prune the search space in the second phase.

The Segment Phase first searches for short patterns containing no gaps, called segments. This phase is efficient. The Pattern Phase searches for long patterns containing multiple segments separated by variable length gaps. This phase is time consuming. The purpose of two phases is to exploit the information obtained from the first phase to speed up the pattern growth and matching and to prune the search space in the second phase. We evaluate this approach on synthetic and real life data sets.

## 5.2 First Phase: Segment-Phase Algorithm

1. Select the string for which a pattern is to be matched. [from the database like protein, nucleotide, DNA egi ATCG abcdacbc …]

2. Find the location of the pattern with the help of function [ indexof(str)] [ user enters the pattern egi abc which is to be searched within the database]

3. Display the location [say L1] and store it in a variable temp [ initially temp = 0 and Display → temp + L1] it displays the locations of the pattern like abc occurs at position or location 1,5,7 etc.

4. Find the substring of the original string starting with index of [ L1 + length of the pattern ] and also increment the counter by 1 [ count = count +1 ]

5. Repeat step2 till the length of substring becomes less than the length of pattern. [ if we are searching abc then continue till the length of substring becomes 2]

6. Return the value of count. [ finally how many times it occurs in a data base]

## 5.3 Second-Phase Algorithm: Wild Pattern Matching

1. Select the string for which a pattern is to be matched. [Searching from the database]

2. Find the last character of the wild pattern [ ab* or abc? With the help of (length –1) ] of string

3. Case 1: If wild-pattern (wp) = '?' then

   i. Find the location [ say L1] of pattern [P] excluding the last character [ left last character * or ? ] in the string S

   ii. Extract the substring starting with the location L1 and of length [pattern length P ] +1 such as ab@ , ab# , abx ie first two occurrences of pattern in ab? And any third character in the sequence.

   iii. Display the substring [as above ab# , ab$ etc.]

   iv. Now find the substring (s) of the original string (S ) with the starting location L1 + length of the pattern (P) and increment the counter by one.

   v. Repeat step first of case 1 till the length of the strings becomes less then the length of the pattern (P)

Case 2: if wild-pattern (wp) = '*' then

[ display entire string starting with ab in case of ab* ]

i. Find the location [ say L1] of pattern [P] in the string [S]

ii. Find the substring of the string with the starting location L1

iii. Display the substring and increment the counter by 1

4. If the pattern is not found then display a "NOT FOUND MESSAGE' otherwise return the value of count

## 5.4 Techniques for comparison

Here we review the techniques for sequence based pattern finding and comparison, and show how these can be extended to RNA structures and abstract representations of protein structure at the fold level. We discuss the deterministic patterns over sequences and distinguish pattern matching from string comparison.

### 5.4.1 Searching Approach

Given a particular target sequence structure in which we are interested, and about which we are lacking certain information, we often wish to find homologous sequences/structures [6, 10, 16] in order to make some hypotheses about the function of that sequence/structure. In general we will have access to a set of reference sequences/structures, which are suitably annotated, with organism of provenance, biological function(s) etc., and which may be grouped into *families* according to certain criteria, e.g. biological function or phylogenetic relationship.

The reference sets may be very large, e.g. all known nucleotide or amino-acid sequences – 16 million or 100,000 records respectively, or all publicly available protein structures – 17,000 The task is thus to use some effective method to relate the target to the reference set, i.e. to perform a search with the target, where effectiveness is measured both by biological usefulness of the results as well as 'speed' of operation. In principle there are two main approaches to searching

**(1) Pair-wise compare the target with each member of the reference set,**

**(2) Group the reference set into families, extract common features of each family, and to match the target with these common descriptions.**

In each case, we will need to *rank* the results in some way in order to be able to consider the most significant. The two approaches can be regarded as being the same if each member of the reference set forms one singleton family.

However, in general, the advantage is that each family usually comprises several members and thus there are fewer families than the reference individuals, and hence less matching operations have to be made than comparison operations. Moreover, matching may be faster than comparison, depending on the detail and form of the common descriptions. The disadvantages are firstly how the choice is made to form family groupings, and how characteristic are the common descriptions. Of course, the groupings and generation of common descriptions is usually performed infrequently, and is certainly not carried out each time a search is made.

### 5.4.2 Comparison of sequences

If we compare a (new) sequence or structure with another sequence or structure, then we can obtain a measure of distance [14,16] or similarity between the two objects; the distance measure should ideally be a metric, i.e.

- Distances should be positive and the distance from an object to itself should be zero
- Distances should be symmetric
- Distances should respect the *triangle inequality* (the direct distance is the shortest distance between two objects) Comparison of two sequences/structures should also produce a set of largest common subsequences or sub-structures

(LCS) - it is not guaranteed that the set is a singleton - and a correlation between the two sequences/structures and each LCS.

The sequences/structures can then be aligned using the LCS. In fact, pair-wise comparison can generalized to *n* objects, although the complexity of a naïve implementation of n-wise comparison can be very high. A form of n-wise distance can be obtained by computing the mean of all the pair-wise distances between the *n* objects.

In general comparison is a more expensive operation than deterministic matching, and more closely related in complexity to probabilistic matching. The most common use of comparison is to pair-wise compare a new sequence or structure *s* with the members of a set *T* of sequences or structures, each of which has some known biological attributes (function, or at least organism of provenance) .

As with probabilistic matching, the result of the comparisons will be the association with each member of *T* of a comparison value for *s*; the task is then to interpret these values. Again, these values can be ordered and also associated with some measure of significance (e.g. E-values or P-values**)**. Thus the comparisons can be ordered, and only those deemed to be significant considered.

### 5.4.3 Characteristics of Sequences

There are many terms used to describe common similarities between sequences or structures, for example pattern, motif, fingerprint, template, fragment, core, site, alignment, weight matrix, profile. For our purposes we will regard a *pattern* as a description of some properties of a sequence [7,8] or structure, and a *motif* as a pattern associated with some biological meaning.

Moreover, if in a new sequence we detect the presence of a pattern known to be characteristic to a certain family, then we can hypothesise that the new sequence belongs to that family, even if we do not know its biological properties yet. In this way patterns may be used for the classification of bio-sequences and for predicting their properties.

**Diagnostic:** A pattern is said to be *diagnostic* for a family if it matches all the known sequences in the family, and no other known sequences. In general, patterns may be characteristic (match most of the sequences in a family and few other sequences), or classificatory (used to decide to which family a sequence belongs).

**Deterministic:** Patterns can be deterministic, i.e. can be used to decide if a sequence or structure matches the pattern or not.

**Probabilistic:** when a value can be assigned to the match.

For instance Cx(2,4)-[DE] is a sequence pattern matching any sequence containing a substring starting with C followed by between two and four arbitrary symbols followed by either a D or an E. Examples of probabilistic patterns are profiles and Hidden Markov Models. Deterministic patterns are simple and pure mathematical concepts, and are easier to interpret than probabilistic patterns; they are also easier to discover from scratch, especially if the data is noisy (contaminated). On the other hand, probabilistic patterns have more modeling power since they permit weights to be attributed to alternatives.

More generally, we will often wish to classify a new sequence or structure *s* using a library or set of *M* motifs. If the motifs are deterministic then matching *s* against the members of *M* will result in a subset of motifs which may be diagnostic for *s*. If *M*

comprises probabilistic patterns then the result of the matching will be the association with each pattern of a match value for *s*; the task is then to interpret these values.

It is usual to associate an ordering relation with match values, for example total ordering over integers or reals, and also to associate some measure of significance with match value, for example E-values (expectation values) or P-values (probability values).

Thus the matches can be ordered, and only those deemed to be significant considered. The use of such motif libraries is predicated on the prior identification of meaningful families, the selection of (possibly representative) family members, and the ability to generate patterns (either by hand, or automatically), which are sufficiently characteristic of the family.

### 5.4.4 Protein family analysis using patterns

Thus, the general protocol [7, 11] for family analysis is used for the following:

(1) Collect sequences (structures) into a family based on biological function or phylogenetic relationships

(2) Make family description by local multiple alignment, global multiple alignment or pattern discovery

(3) Use the description to identify more family members

(4) Analyze the extended set to see if the members are biologically related to the original family members

### 5.4.5 Regular expressions

Notation used for regular expression:

**Symbol**: for each symbol **a** in the alphabet of the language, the regular expression **a** denotes the language containing just the string a

**Alternation**: Given 2 regular expressions M and N then M | N is a new regular expression. A string is in language (M|N) if it is language M or language N. The language (**a**|**b**) = {a,b} contains the 2 strings a and b.

**Concatenation**: Given 2 regular expressions M and N then M•N is a new regular expression. A string is in language (M•N) if it is the concatenation of two strings □and □such that □is in language M and □is in language N. Thus the regular expression **(a|b)•a** = {aa, ba} defines the language containing the 2 strings aa and ba

**Repetition**: M* stands for zero or more times repetition of M, M+ one or more times, and M? for zero or one occurrences of M.

**Character ranges:** [a-zA-Z] character set alternation, '.' any single character except a new-line (i.e. a wild card)

### 5.4.6 Expressions for biosequences

In general we have the following basic alphabets:

- □= {a, t, c, g} for DNA nucleotides,
- □= {a, u, c, g} for RNA nucleotides.
- In the case of proteins we have a 20 character alphabet of amino acids □= {A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}.

## 5.5 Structural sequence patterns

Eidhammer introduce CBSDL, a constraint-based structure description language, where *structural* patterns contain constraints on the [5,10,11]

(1) Length of a substring to match a specific component;

(2) Distance (in the input string) between substrings to match the different components of a pattern;

(3) Contents of a substring to match a component, e.g. the second symbol should be an a or a t;

(4) Positions on the input string where a particular component can match;

(5) Correlation between two substrings matching different components, e.g. the substrings should be identical, or the reverse of each other.

This definition thus includes purely sequential patterns, which do not include a correlation constraint and are within the class of regular languages, for example PROSITE patterns. Structural patterns have at least one correlation constraint, for example repetitions or palindromes, and may describe context-free languages or even languages beyond the expressive power of context-free grammars, although there may be some languages in these classes, which they cannot describe.

From the point of view of bioinformatics, sequential patterns can thus describe sequences, whereas structural patterns can describe 'folded' strings, i.e. RNA structures such as stem-loops and pseudo-knots, and some topological descriptions of protein structures. Some examples of patterns which can be described by this language are shown below. We use Greek letters to indicate sub-patterns, possibly superscripted by *r* for reverse and *c* for complement, and underline corresponding sections of sequences.

### 5.5.1 Example of structural sequence patterns

Tandem repeat □-□acg-acg

Simple repeat □-□-□acg-aaa-acg

Multiple repeat □-□-□-□-□acg-aa-acg-uu-acg

Palindrome □-□r acg-gca

Stem loop □-□-□rc acg-aa-cgu

Pseudoknot □-□1-□-□2-□rc-□3-□rc augg-cuga-aggc-cgau-c-ucag-ggcau-aucg-ccgu

### 5.6 Learning – pattern finding

Brazma [12,16] have surveyed pattern discovery in biosequences and in the following we generalize their definitions to biostructures as well as biosequences.

A *protein family* F+ is a set of protein sequences or structures sharing definite functional or structural properties. If we have a *language L* of strings or structures then F+ is a subset of the total set of all possible sequences/structures that can be generated by the grammar of *L*, and F- are all those sequences/structures in *L* which do not belong to the family.

Pattern finding is then the problem of automatically finding functions approximating the characteristic function for the family F+. An algorithm for solving this problem takes as input a *training set* consisting of *positive examples*, which are sequences from F+, and optionally *negative examples* which are sequences from F-. This is a machine learning problem, namely that of learning a general rule from a set of examples. When both positive and negative examples are given, it is called the *classification problem*, and when only positive examples are given, it is called the *conservation problem*.

In the classification problem, we are given a set of sequences/structures S+ believed to be members of a family F+, and a set S- of sequences/structures believed not to be members of F+, i.e. S+ □F+ and S- □F-. We also assume that F+ and F- are disjoint.

The goal is then to find *compact* "explanations" of known sequences, i.e. functions that return true for all s ∈ S+ and false for all s ∈ S-, and have a high likelihood for returning true for s ∈ F+ and false for s ∈ F-. Furthermore, we would like to try to predict the family relationship of yet unknown sequences.

# Chapter 6

# Techniques for Biological Database Searching

> *"This project brings us back to the beginning of time ... and the origin of life – dealing with DNA, RNA, proteins, human genome, etc. The goal is to apply database technologies for genome sequencing, indexing and searching."*

## 6.1 Introduction

This project falls under the broad area of study called *Bioinformatics* or *Computational Molecular Biology* [9,14,16]. Among the various fields of bioinformatics, we are particularly interested in efficient searching and comparison of the biological data. These tasks form a basic part of bioinformatics, and make

contributions to many application areas in bioinformatics such as data analysis, data mining, biological reasoning, and evolutional deduction.

In this chapter, we are going to study the existing methods of biological database searching, and investigate their respective advantages and disadvantages. Then, we develop our own system that can hopefully provide the optimal solutions in database searching – at least for certain biological applications, even if it cannot afford the optimal solutions for all.

After this project is finished, its results would help the domain experts (biologists and doctors) in many ways. For example, it can be used in analyzing the functionality of a gene, in locating and curing of a disease causing gene, in deducing the process of evolution – to name a few. Because all of these tasks basically involve data searching and comparison, with the help of our efficient techniques, they would be able to perform these tasks efficiently. They can enjoy quick response times without having to use very expensive hardware resources.

Finally, the results of this project may help the biologists, to certain extent, in achieving their ultimate goal, i.e., to decode the entire language of instructions of the nature (or God) used in creating and activating all the living things.

## 6.2 Molecular Biological Background

All organisms (living things) possess the discrete entities called *genes* that are the basic inherited units of biological function and structure [6,10]. An organism inherits its genes from its parents, and relays it own genes to its offspring. The study of heredity and variations of the organisms is called *Genetics*.

Once the concept of gene was a logical one. But in the later half of the $20^{th}$ century, the physical mechanism of the gene can be determined and studied with the help of

*Molecular Biology*. Molecular biologists determined that the gene is made of DNA (deoxyribonucleic acid) – that is, DNA is the heredity material of all species. *Crick* and *Watson* determined in 1953 that the structure is a *double helix* and concluded correctly that this specific form is fundamental to DNA's function as the agent that stores and transfers genetic information.

The DNA double helix is elegant and simple. Each strand of the DNA double helix is a polymer (a huge compound made up of small simple molecules) consisting of four elements called *nucleotides*: *A*, *T*, *C*, and *G* (for adenine, thymine, cytosine, and guanine). The two strands of DNA are perfectly complementary.

When a *T* resides on one strand, an *A* occupies the corresponding position on the other strand; when there is a *G* on one strand, a *C* occupies the corresponding position on the other. That is, *T* pairs with *A*, and *G* pairs with *C*. (These pairs are sometimes referred to as *base-pairs*.) This redundancy is useful at the time of cell division. A complete set of genetic information is passed to each daughter cell.

The DNA double helix unravels, and each strand serves as a completely sufficient template upon which a second strand can be synthesized. In addition, the redundancy also provided great resiliency against loss or damage of information during cell life.

From a computer scientist's point of view, the DNA double helix is a clever, robust information storage and transmission system. Like the binary alphabet {0, 1} used in computers, the four-letter alphabet of DNA {*A*, *T*, *C*, *G*} can encode messages of arbitrary complexity when encoded into long sequences.

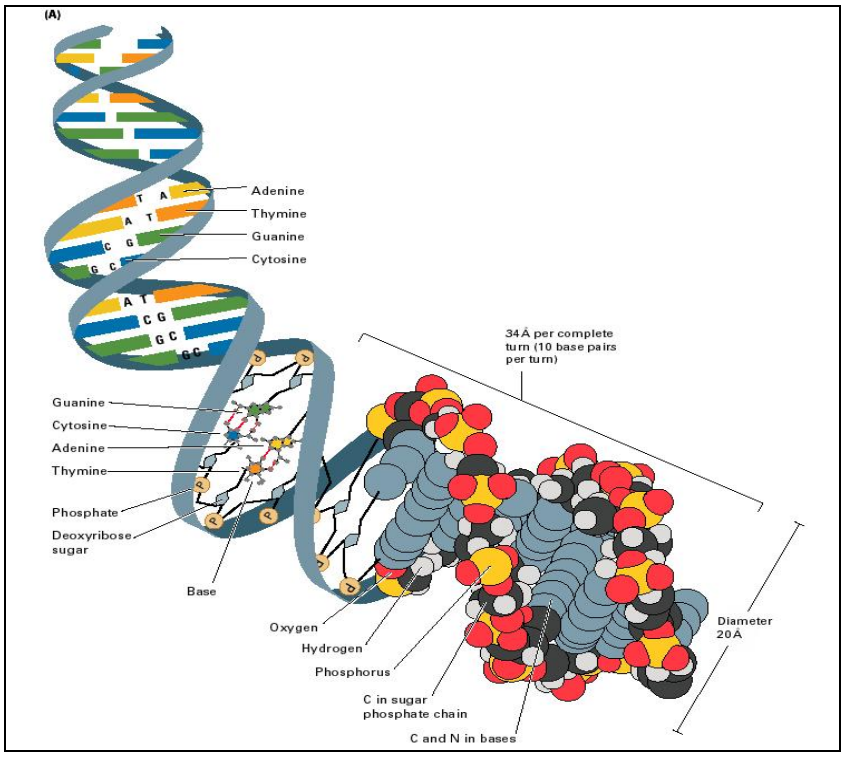



Figure 1. Structure of DNA double helix

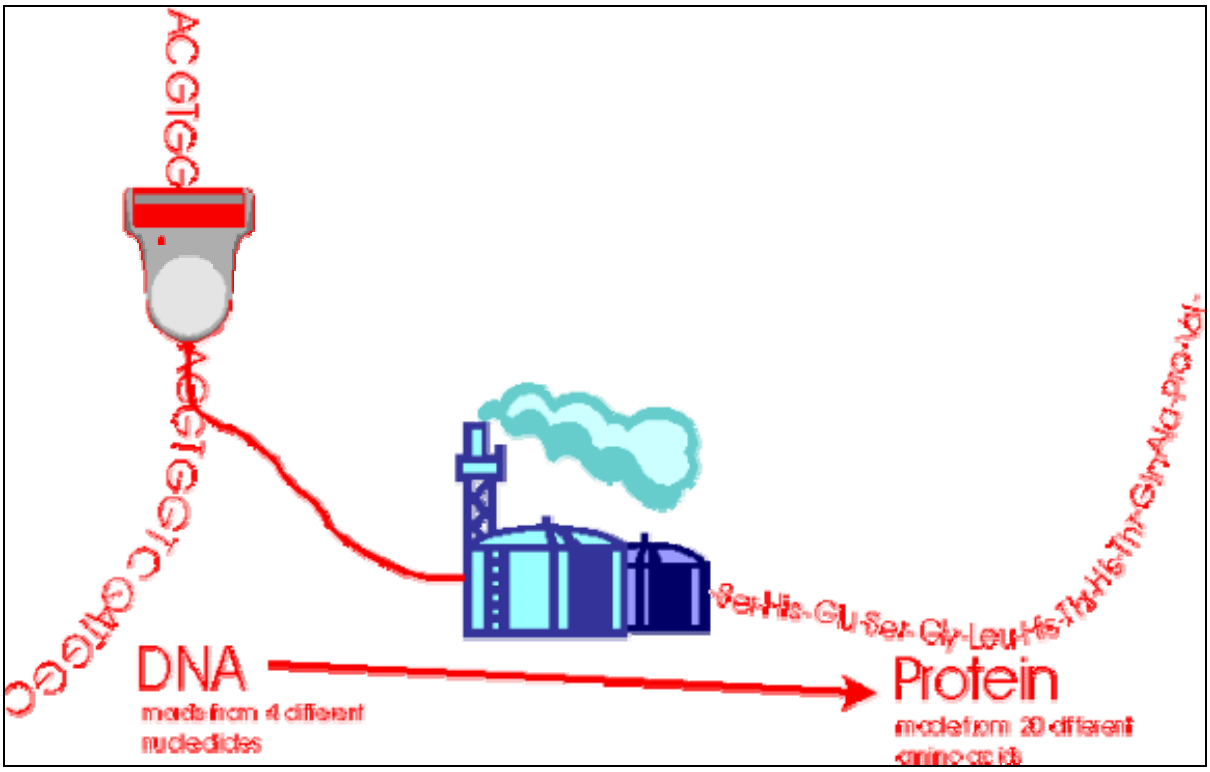
Particular stretches of the DNA are copied directly into an intermediate molecule called RNA (ribonucleic acid, also composed of *A*, *T*, *C*, and *G*). RNA is then translated into a protein – which is again a linear chain assembled from the 20 different amino acids. Each consecutive [9,14] triplet of DNA elements specifies one amino acid in the protein chain.

In this fashion, biology "reads" DNA (actually, the RNA copy of the DNA) as if it were a Turing machine tape. Once synthesized, the protein chain folds according to the laws of physics into a specialized form, based on the particular properties and the order of the amino acids. The structures of a protein can be viewed as a hierarchy:

- Primary structure (linear amino acid sequence)

- Secondary structure (local sequence elements with well determined regular shape)
- Tertiary structure (3D structure of whole sequence)
- Quaternary structure (combination of proteins)
  - Motif (combines a few secondary structure elements with a specific geometric arrangement)
  - Domain (combines several secondary structure elements and motifs; has a specific function)

**Figure 2. Synthesis of protein from DNA (*below*):**



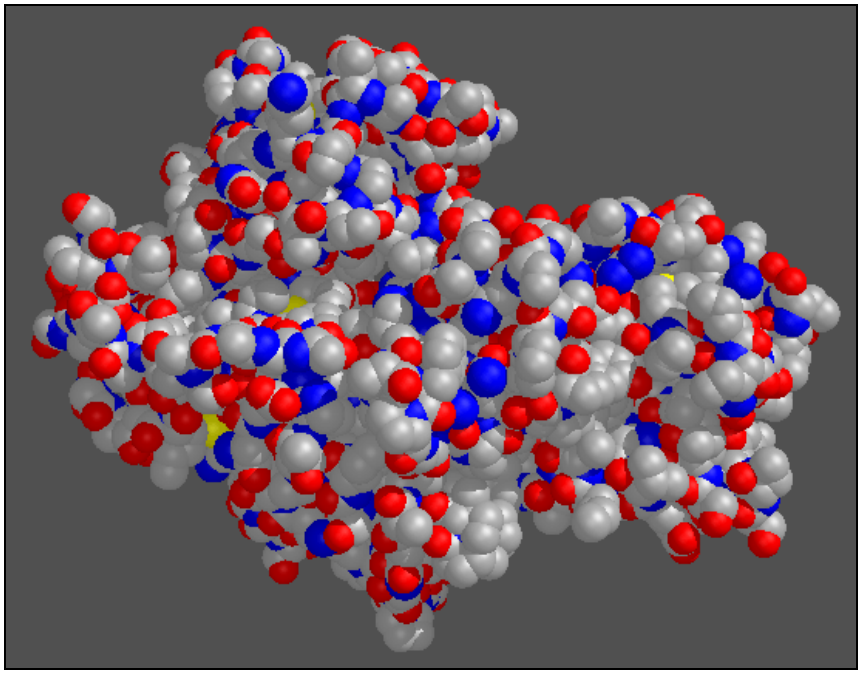



Figure 3. Three-dimensional structure of a protein

The human genome (the totality of genetic information in each person) contains about 3 billion nucleotides.

These are distributed among 23 separate strands called chromosomes, each containing about 50 to 250 million nucleotides. Each chromosome encodes about 10 to 50 thousand genes. Not all the sequences in the DNA are genes. Only certain parts have their particular functions to serve as genes. Genes spell out the instructions for making proteins and controlling their production. But, on the other hand, many different DNA sequences can encode the same gene.

**Goal of Molecular Biology**

A major goal in molecular biology is functional genomics, or the study of the relationships among genes in DNA and their function. Gene function can be viewed through several prisms. A common interpretation [6,10] is that function describes the role of a gene product, usually a protein, in reacting with other proteins in a metabolic or signaling pathway.

However, molecular biologists know that protein interactions are dependent on protein structure, or shape. Functions can also be conveyed through annotations written by researchers who have studied in detail a given protein and its interactions with other proteins. The notion of function is essentially related to protein shape and to the behavior of the organs that make up a living being; for example, the study of cell differentiation in original stem cells is related to cell function.

Biologists and computer scientists may conclude that the ultimate objective of functional genomics is: Given the DNA of an organism, produce a simulator for a cell of that organism. That simulator (or flowchart representing metabolic and signaling pathways) embodies all that it knows about a cell's behavior, allowing in-silico

experiments that enable biologists to bypass costly and ethically sensitive in-vitro or in-vivo trials. We are far from this goal, but it is an area where computer science can provide considerable research impetus.

**Biologists deal with essentially four types of data structures:**

- Strings  To represent DNA, RNA, and sequences of amino acids;
- Trees To represent the evolution of various organisms;
- Sets of 3D points and their linkages. To represent protein structures; and
- Graphs. To represent metabolic and signaling pathways.

Furthermore, biologists are often interested in substrings, subtrees, subsets of points and linkages, and subgraphs. Strings (such as words and phrases) are also used to express annotations that convey a meaning given by researchers, though such meanings are sometimes vague and incorrect. Biological data is often characterized by huge size, the presence of laboratory errors (noise), duplication, and sometimes unreliability.

For inferring function from the existing data, a biologist must consider three factors:

- Genes, or substrings of DNA capable of generating proteins;
- Protein structures represented in 3D space; and
- The roles of these proteins within metabolic and signaling pathways.

## 6.3 Biological Databases

In the field of bioinformatics, a large number of databases are created and stored by a large number of organizations. Generally, there are five different types of databases.

1. **Sequence databases**
2. **Structure databases**
3. **Map databases**
4. **Model organism databases**
5. **Bibliographic databases**

Among these database types, sequence databases and structural databases are mostly related to this project. The growth rates of these databases are very fast. The amount of data doubles in less than a year [ 14,16]

### 6.3.1. Sequence Databases

There are two types of sequences stored in the sequence databases:

- Nucleotide or DNA sequence (made up of a four-letter alphabet)
- Amino acid or protein sequence (made up of a twenty-letter alphabet)

The most popular DNA sequence databases are:

- GenBank (NCBI – National Center for Biotechnology Information, USA)
- EMBL (European Molecular Biology Laboratory)
- Nucleotide Database
- DDBJ (DNA Data Bank of Japan).

The most popular protein sequence databases are:

PIR (Protein Information Resource by NBRF – National Biomedical Research Foundation, USA)

- SwissProt (SIB – Swiss Institute of Bioinformatics), and TrEMBL (EMBL).

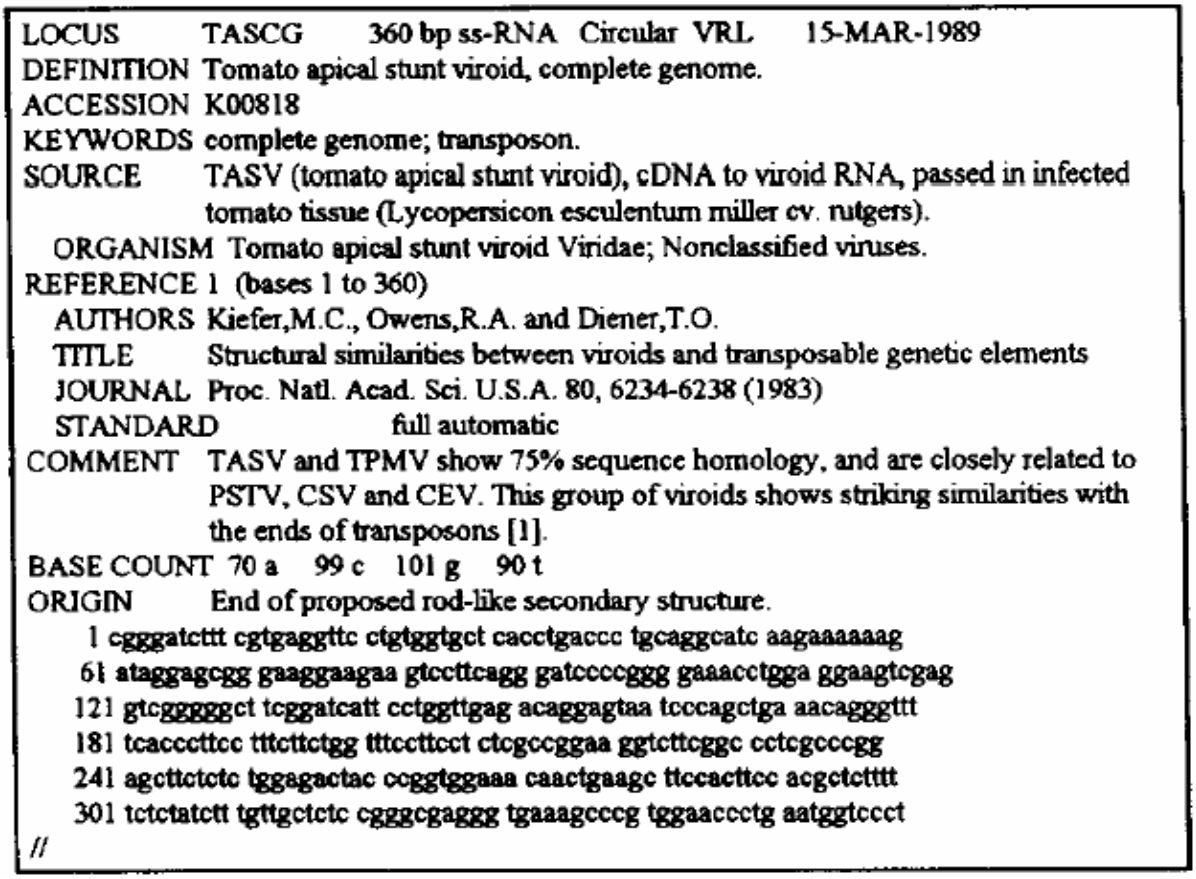
The above-mentioned organizations conduct experiments to collect their data sequences. In addition, other various research organizations also contribute data to them.

They also exchange and share the data among them. Some of the data stored in these databases are primary ones (i.e. directly obtained from the experiments), but some are derived ones (i.e. obtained by analysis, deduction and prediction of the primary data). These data (both primary and derived) are static or archive in nature. But there may be some corrections or updating on them if the original sequences seem to be incorrect (because of an inaccurate experimental method, for example).

According to the researches to date, data sequences (both DNA and protein) are known to be pseudo-random in nature, and no deterministic pattern can be found in them.

Since the databases are developed by different organizations, they have different purposes, schemas, storage structures and access methods, etc. However, most of the databases store the sequences in flat file format (although their schema may be different).

The following figure shows a sample entry of GenBank database [9,12].



```
LOCUS       TASCG      360 bp ss-RNA  Circular  VRL     15-MAR-1989
DEFINITION  Tomato apical stunt viroid, complete genome.
ACCESSION   K00818
KEYWORDS    complete genome; transposon.
SOURCE      TASV (tomato apical stunt viroid), cDNA to viroid RNA, passed in infected
            tomato tissue (Lycopersicon esculentum miller cv. rutgers).
  ORGANISM  Tomato apical stunt viroid Viridae; Nonclassified viruses.
REFERENCE   1  (bases 1 to 360)
  AUTHORS   Kiefer,M.C., Owens,R.A. and Diener,T.O.
  TITLE     Structural similarities between viroids and transposable genetic elements
  JOURNAL   Proc. Natl. Acad. Sci. U.S.A. 80, 6234-6238 (1983)
  STANDARD                full automatic
COMMENT     TASV and TPMV show 75% sequence homology, and are closely related to
            PSTV, CSV and CEV. This group of viroids shows striking similarities with
            the ends of transposons [1].
BASE COUNT  70 a   99 c   101 g   90 t
ORIGIN      End of proposed rod-like secondary structure.
        1 cgggatcttt cgtgaggttc ctgtggtgct cacctgaccc tgcaggcatc aagaaaaaag
       61 ataggagcgg gaaggaagaa gtccttcagg gatccccggg gaaacctgga ggaagtcgag
      121 gtcgggggct tcggatcatt cctggttgag acaggagtaa tcccagctga aacagggttt
      181 tcacccttcc tttcttctgg tttccttcct ctcgccggaa ggtcttcggc cctcgcccgg
      241 agcttctctc tggagactac ccggtggaaa caactgaagc ttccacttcc acgctctttt
      301 tctctatctt tgttgctctc cgggcgaggg tgaaagcccg tggaaccctg aatggtccct
//
```

Figure 4. A sample data entry of a virus in GenBank

For sequence searching and comparison purposes, most of these database systems use BLAST (Basic Local Alignment Search Tool), FASTA, and SSEARCH (Smith-Waterman Search). In addition, there are various other tools for data analysis, data mining, and prediction, etc.

### 6.3.2 Structural Databases

The tertiary and quaternary structures of the proteins are three-dimensional in nature. There structures are stored in structural databases. The typical example is PDB (Protein DataBank by RCSB – Research Collaboratory for Structural Bioinformatics).

PDB has its proprietary data file format. In this format, 3D data are stored as the points in a 3D coordinate system.

It provides facilities for viewing the 3D data by means of static images or by means of the graphical techniques such as VRML (Virtual Reality Modeling Language). It also provides the facilities for comparing and analyzing the 3D data. Such tools include VAST (Vector Alignment Search Tool), SCOP (Structural Classification of Proteins), FSSP (Fold Classification based on Structure-structure alignment of Proteins), etc.

### 6.3.3 Collaboration of Databases

Since the biological databases are heterogeneous in nature, an effort of collaboration is required in order to maintain the integrity and interoperatibility among them. This is particularity important if an application needs to use the data stored in more than one database. The degree of collaboration may vary from using the client-server architecture to transfer data between distributed heterogeneous databases to creating a seamless homogeneous database.

Since 1990s, a set of agreements- protocols has been made between the major research institutes such as NCBI, EMBL, and DDBJ. These protocols include daily exchange of data, standardizing formats, standardizing rules, etc. But more collaboration efforts are still required. These include:

- Global standardization of protocols, rules, and formats.
- Distributed computing over multiple heterogeneous databases on the web.
- Upgrading databases from flat files to real database files, etc.

Some of the attempts on these issues are:

- CORBA based approach by LSR (Life science Research Domain Task Force) of OMG (Open Management Group).

- Java based approach by BioWidget.

Today, some database systems are offering web-based application facilities. This can be viewed a step forward to the goal of total database collaboration. ([NCBI, 2001].

The following picture shows a web-based searching tool of NCBI.

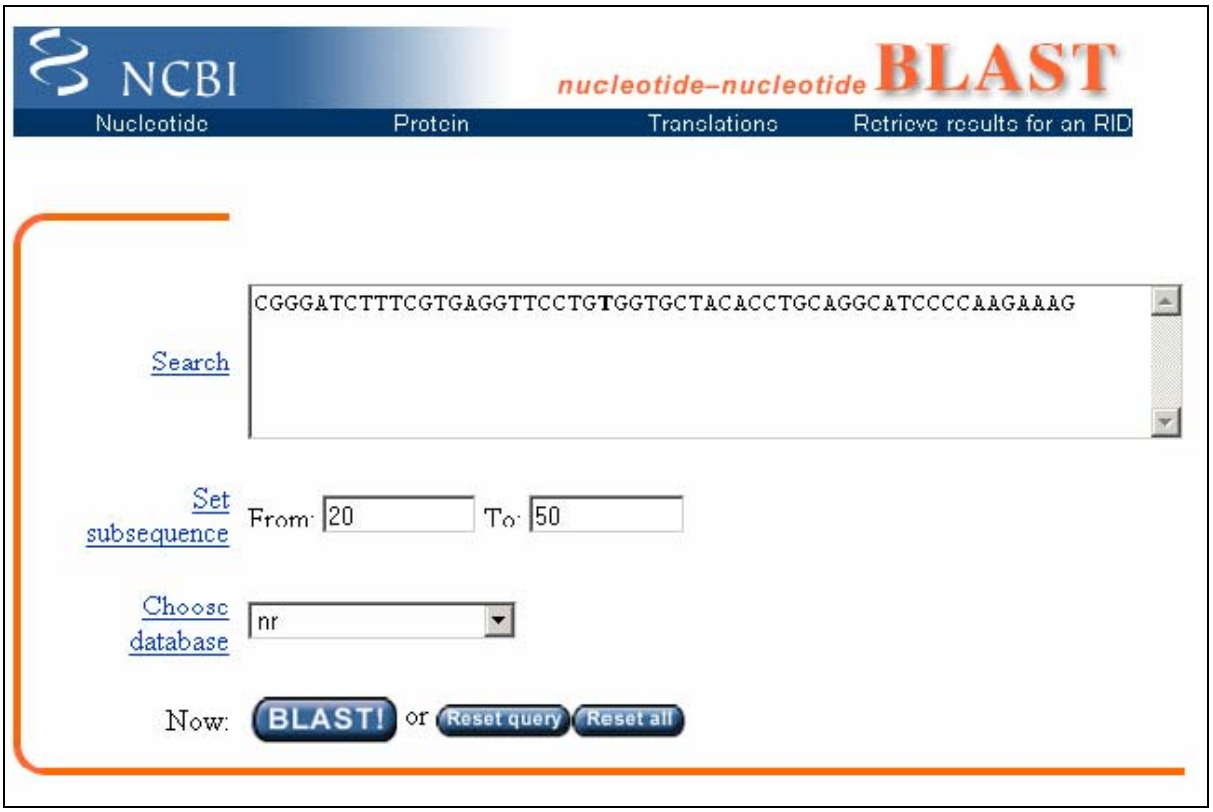



Figure 5. A search page from NCBI BLAST website

## 6.4 Sequence Matching

The general definition of "sequence matching" [10, 16] is that given a query sequence, the sequences in the database are search through in order to find ones that "match" the

query sequence. The **terms matching, comparison, searching, querying, and alignment are generally interchangeable.**

As mentioned above, we have very large of biological databases which are growing rapidly, and so we must have "smart" algorithms for this purpose.

Strictly speaking, sequence matching may not be very useful in its own. But it plays an important role in many other bioinformatics tasks such as sequence analysis, gene mapping, structural prediction, data mining, reasoning and deduction, etc.

For example, we can deduce the function of an unknown protein sequence by comparing it to a set of known ones already stored in the database. The usefulness of sequence matching in bioinformatics can be compared with that of ordinary searching and sorting algorithms which are useful in many other areas of computing.

Basically, there are two types of sequence matching:

- Exact matching (the source and target strings must be exactly the same) and
- Approximate/ similarity matching (the source and target strings must be similar according to some predefined criteria).

According to the nature of the biological applications, the latter is usually more useful.

Here, the computer scientists try to use the existing general pattern matching techniques in biological sequence matching task. But some techniques (e.g. sequential search) cannot be used due to the hugeness of the biological data. Some techniques cannot build the data structures bigger than certain sizes due to the memory bottlenecks, etc.

So the computer scientists invent the new algorithms dedicated to biological sequences (e.g. BLAST – Basic Local Alignment Search Tool [NCBI, 2001]), modify the existing algorithms to make them adaptable to biological sequences (e.g. suffix trees [Hunt et al., 2001]), or adopt the ideas from other areas to solve the problems in biological sequence searching (e.g. wavelets [Kahveci, Singh, 2001]).

However, since biological sequences have no words in them, the pattern matching algorithms based on words cannot be used.

### 6.4.1 Parameters of Sequence

The followings are the parameters common to all sequence matching techniques.

- **Source (S):** A sequence (string) specified by a user in his query. It is sometimes called pattern. (|S| = m)

- **Target (T):** A sequence (string) in the database. One that a user wants to match against the source. It is sometimes called text. (|T| = n)

- **Similarity score (SS):** Let X and Y be two different strings. The similarity score between X and Y is defined by: the score for matches *minus* the score for mismatches *minus* the score of gaps. The score for each kind is calculated as the frequency of this kind *times* a pre-specified weight. The weights may be different from scoring system to system (e.g. PAM and BLOSUM). Please refer to *Figure 8* to learn the score concept.

- **Edit distance (ED):** Let X and Y be two different strings. The minimum number of operations (insertion *plus* deletion *plus* replacement) required to transform X to Y is called the edit distance of these two strings. For example, the edit distance of the following two strings is 4. Either similarity score or edit distance can be used to measure the similarity between two sequences.


Figure 6. Transformation of the string **ACTTAGC** to **AATGATAG** using edit operations.

(Here the edit operations are represented by I = insert, D = delete, and R = replace.)

- **Threshold score/ distance (σ):** The maximum distance or the minimum similarity score between the source and the target for which a target string is to be shown in the result set of an approximate matching. The user must specify the threshold distance or score.

- **Sensitivity:** Not all the algorithms can provide optimal solutions. Sensitivity is the measure of how many good answers are left out from a query result in the case of a sub-optimal solution. It is calculated as the result size *divided by* the number of good answers.

- **Alignment:** A process of aligning two sequences so as to gain the maximum score or the minimum distance. For example in the following picture, we are trying to get the alignment like one shown in the second case.

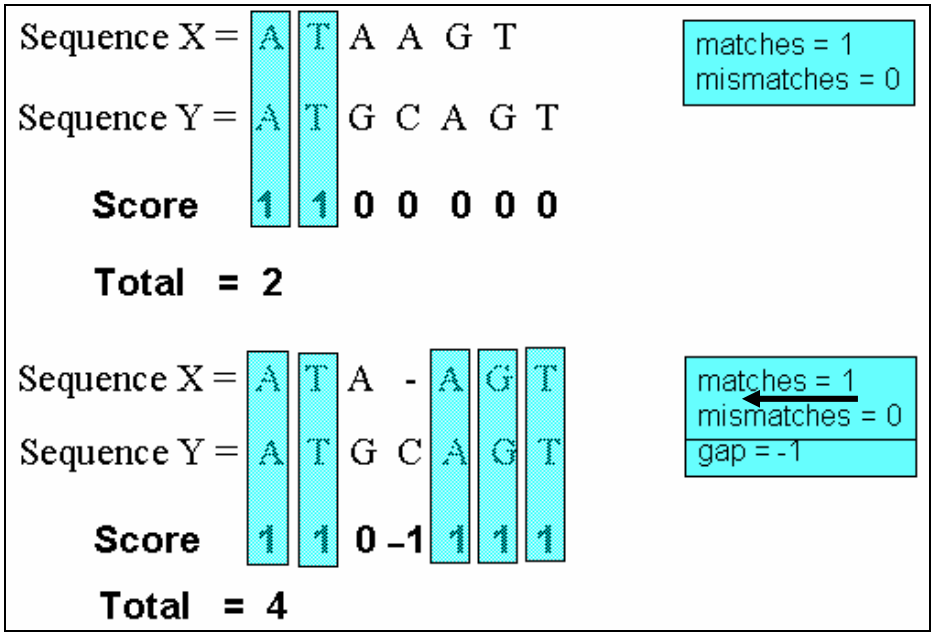



Figure 7. Finding the alignment between two strings to gain the maximum score

- **Pair-wise matching**: Matching between only two sequences at a time. The term "matching" throughout this paper means pair-wise matching unless otherwise stated.
- **Multi-sequence matching:** Comparing more than two sequences simultaneously.
- **Global alignment:** Aligning two entire sequences.
- **Local alignment:** Aligning the subsequences of two given sequences.
- **Substitution Matrices:** The matrices used for comparing the similarity between two characters in the case of approximate matching.

| | A | C | T | G |
|---|---|---|---|---|
| **A** | **1** | **0** | **0** | **0** |
| **C** | **0** | **1** | **0** | **0** |
| **T** | **0** | **0** | **1** | **0** |
| **G** | **0** | **0** | **0** | **1** |

Figure 8. Substitution matrix for nucleotides in DNA sequences

### 6.4.2 Sequence Matching Methods

In this report, we are going to discuss the some of the attempts to solve the problem of sequence matching [12,14,16].

Generally, the sequence matching methods can be classified as:

- Dynamic Programming methods
- Heuristics methods
- Indexing methods (conventional indexing and metric space indexing)

#### 6.4.2.1 Dynamic Programming Methods

Use of Dynamic Programming (DP) is the earliest attempt to solve the sequence-matching problem. DP algorithms can provide the optimal (100% sensitive) solution. But the disadvantage is their slowness.

A DP algorithm uses a bottom-up approach. It starts by solving the smallest problems, and uses the partial solutions to solve bigger and bigger problems. It uses extra memory called the similarity matrix to store the intermediate values.

Two popular DP algorithms are:

- Needleman-Wunsch algorithm (for global alignment)
- Smith-Waterman algorithm (for local alignment)

#### 6.4.2.2 Heuristics Methods

Heuristics methods trade speed for precision. They can only provide sub-optimal solutions in which some good answers may be left out. But, as mentioned above, the sub-optimal solutions are still very useful for the biologist. So, heuristics methods are widely used for searching large biological database.

However, indeed, these methods partially use DP inside them. It is very important that the similarity results obtained must be statistically significant rather than just coincidences. Scoring systems and substitution matrices are used to ensure this. Those results that are statistically insignificant are thrown out since the initial steps.

Today, heuristics methods are largely used in the major research institutes. The two industrial *de facto* standards are:

- BLAST (Basic Local Alignment Search Tool) (Altschul et. al, 1990)
- FASTA (Pearson, 1985)

**Example: BLAST**

The idea of BLAST is to integrate the substitution matrix in the first stage to find the hot spots (very high similarity regions). Here we need to define some fundamental objects concerning BLAST.

Given two strings X and Y, a *segment pair* is a pair of equal length respective substrings of X and Y, aligned without gaps.

A *locally maximal segment* is a segment whose alignment score (without spaces) cannot be improved by extending it or shortening it.

A *maximum segment pair* (MSP) in X and Y is a segment pair with the maximum score over all segment pairs in X, Y.

When comparing all the sequences in the database against the query, BLAST attempts to find all the database sequences that when paired with the query contain an MSP above some cutoff score S. Such a pair is called hi-scoring pair (HSP). S is chosen such that it is unlikely to find a random sequence in the database that achieves a score higher than S when compared with the query sequence.

The stages in the BLAST algorithm are as follows:

- Given a length parameter w and a threshold parameter T, BLAST finds all the w-length substrings (called words) of database sequences that align with words from the query with an alignment score higher than T. Each such hot spot is called a hit in BLAST.

- Extend each hit to find out if it is contained in a segment pair with score above S (HSP).

The first stage may be implemented by constructing, for each w-length word $w_i$ in the query sequence, all the w-length words whose similarity to $w_i$ is at least T. These words are stored in a data structure which is later accessed while checking the database sequences.

It is usually recommended to set the parameter w to values of 3 to 5 for amino acids, and ~12 for nucleotides.

#### 6.4.2.3 Indexing Methods

Dynamic Programming and heuristics techniques do not build any persistent data structure. They build the data structures required for them on the fly while executing the program. In the indexing approach, the idea is to construct a persistent data structure (an index or a similar one) on the database sequences in advance. If properly constructed, this will surely enable first database searching.

Some researchers adopt the existing general pattern matching techniques using indexes, and modify them for biological sequence matching. Some others propose entirely new approaches for sequence matching. Among all these solutions, some can offer optimal solutions, but some are sub-optimal ones. Other possible issues in indexing may be memory and storage bottlenecks, caching, etc. Most of the techniques are still under research, and are not publicly accepted yet.

We can classify these indexing techniques into two categories:

- **Traditional indexing**
    - Suffix Tree ([Hunt et al., 2001])
    - NFA ([Baeza-Yates, Navarro, 1999])
    - Suffix Array
    - R-tree
    - q-gram, etc.
- **Metric space indexing** (measuring similarity or distance in Euclidean/ metric space)
    - Wavelets ([Kahveci, Singh, 2001])

- GNAT trees and M-trees ([Chen, Aberer, 1997])
- TSVQ ([Giladi et. al, 2000]), etc.

## 6.5 Suffix Tree

Suffix tree is one of the traditional indexing techniques. Suffix tree is basically used for exact matching, but it can also be extended and utilized in approximate matching. The purpose is to build a persistent index for biological sequences.

Suffix trees are compressed digital tries. Given a string, we index all suffixes, i.e. for a string of length 10, all substrings starting at index 0 through 9 and finishing at index 9 will be indexed. The root of the tree is the entry point, and the starting index for each suffix is stored in a tree leaf. Each suffix can be uniquely traced from the root to the corresponding leaf. Concatenating all characters along the path from the root to a leaf will produce the text of the suffix.

An example digital tree of **ACATCTTA** is shown in *Figure 9*. The number of children per node varies but is limited by the alphabet size. This tree can be compressed to form a suffix tree, shown in *Figure 9*. To change a tree into a suffix tree, we conceptually merge each node which has only one child with its child, recursively, and annotate the nodes with the indices of the start and end positions of the substrings indexed by that node.

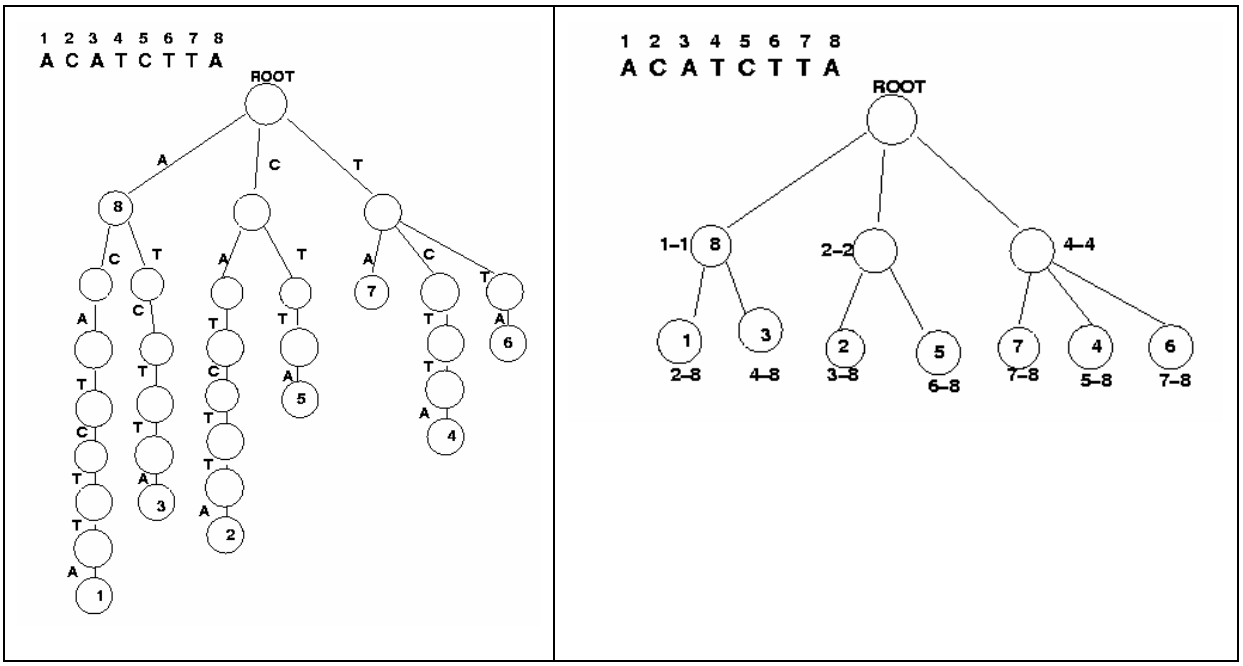



Figure 9. A tree on **ACATCTTA** and suffix tree on the same string

Suffix link is a structure used to reduce the tree building time, but it can cause memory bottleneck. So they avoid the use of suffix links. In addition, multiple passes are performed over the sequence, constructing the suffix tree for a subrange of suffices at each pass.

## 6.6 Methods for implementation of sequence matching

These methods can be implemented in different modes:

### 6.6.1 Sequential computing

Traditionally, sequential implementation is used to implement each of the abovementioned technique. However, today, parallel and distributed computing techniques become very much developed. So, these two are the other options for implementing the sequence searching algorithms.

### 6.6.2 Parallel computing

Parallel computing is a simple implementation for both exact matching and approximate matching [10,16]. It can yield the satisfactory performance without having to scarifying the sensitivity. But the disadvantage is that it requires a lot of resources (parallel homogeneous computers/ CPUs and special hardware devices in some cases). Two basic types of parallelism are:

- Fine grain: all processors cooperate to determine the similarity score. (Suitable for SIMD – Single Instruction and Multiple Data stream architecture.)

- Coarse grain: each processor performs independent comparisons. (Suitable for MIMD – Multiple Instruction and Multiple Data stream architecture.) ([Yap et. al, 1996])

#### 6.6.2.1 Hardware Oriented Approach

Some parallel computing techniques are hardware oriented. The example case discussed here is form the paper [Hoang, 1993]. It utilizes systolic arrays implemented as Splash 2 Programmable Logic Array (PLA).

Its objective is to calculate the edit distance between two strings (a fundamental step in approximate matching), and it uses a Dynamic Programming algorithm for this purpose. The algorithm is implemented using a hardware description language called VHDL. The algorithm used here is very much similar to Needleman-Wunsch algorithm.

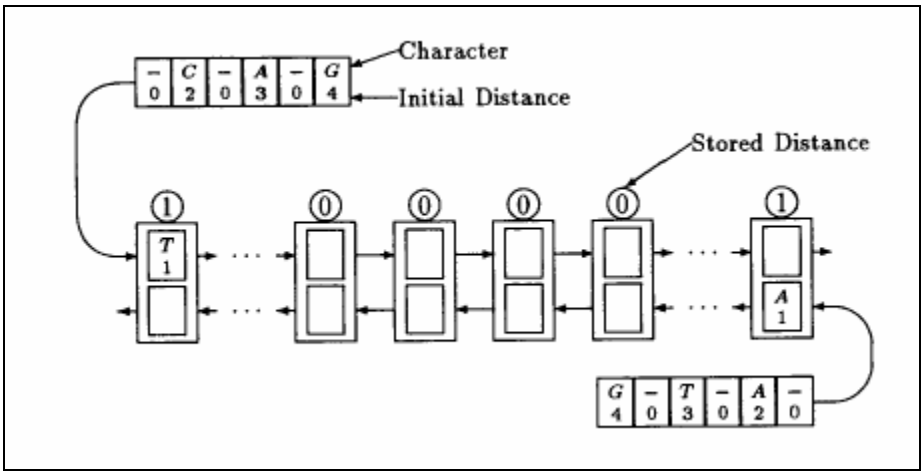



Figure 10. Bi-directional data flow in Splash 2 PLA

*Space complexity*: **O (min(m,n))**

*Time complexity*: basic algorithm **O(mn)**, enhanced algorithm **O($n^2$ / log n)**

#### 6.6.2.2 Software Oriented Approach

Most of the parallel implementations are software-oriented ones. They use conventional multi-processor computers, or a set of standalone processors distributed over a LAN. The sample case discussed here is from the paper ([Matsuda, 1995]).

It implements FASTA algorithm in parallel computing. It uses logic programming for data-parallel approach. Logic programming includes the query capabilities of a relational database with pattern matching operations. It stripes an entire database to map on local disks of workstations, search every partition in parallel, and combines their partial results into a complete answer.

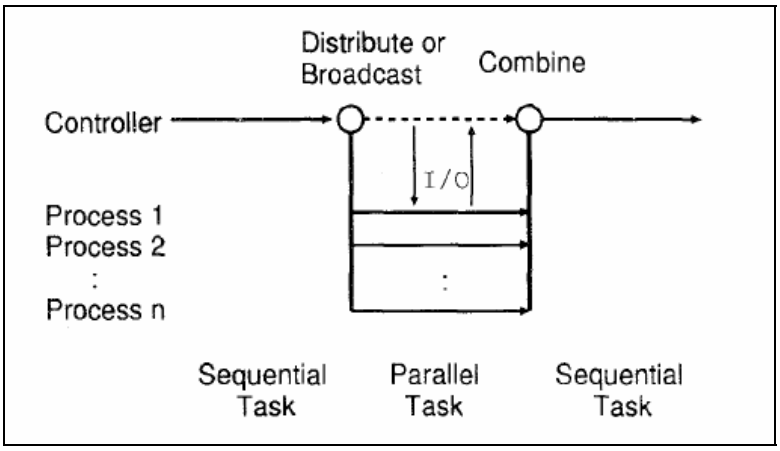



Figure 11. Execution model based on data-parallel execution

There are two types of agents in the system: a *controller* coordinates the entire system, and the processes are supervised by the controller to perform query processing.

There are two ways to implement parallelism.

1. *Distributing input data*: if a user asks a query with a large set of input data, the controller divides the input data then distributes the partial input to each process.

2. *Broadcasting condition*: if a user retrieves a set of data that satisfies specific conditions, the controller separates the entire database into partial blocks, decides the assignment of the blocks to processes, and broadcasts the assignment information to all processes.

The algorithm is implemented by the use of "prolog engine" here. It operates in SIMD (Single Instruction, Multiple Data stream) mode.

Performance: with 40 workstations, it can speed up 18.6 fold over a sequential algorithm for striping and searching, and 35.5 fold for searching only.

### 6.7.3 Distributed computing

Distributed implementation uses multiple heterogeneous processors distributed over a network (LAN or WAN). The sample attempt discussed here is from the paper ([Anderson, Bansal, 1999]).

This architecture is more suitable for applications involving multiple heterogeneous databases (compared to the homogeneous parallel processing). Common Object Request Broker Architecture (CORBA) is used to map multiple processes to a heterogeneous set of architectures and operating systems.

It uses a two-phase process for approximate sequence matching (similar to one used in [Chen, Aberer, 1997]).

In the first phase, BLAST algorithm is used to prune out dissimilar sequences. BLAST is an approximate string matching technique for similarity matching. In the second phase, Smith-Waterman (Dynamic Programming) algorithm is used for more accurate alignment using dynamic matrix technique. This two-phase approach greatly reduces the complexity. In this distributed system, the communication overhead is negligible compared to the string matching time.

CORBA architecture is based on the object-oriented technique. The system is composed of various objects: coordinator object, queue object, server object, and genome object. The genome object constructs a map of genes and their positions in the file. The purpose of the map is to reduce searching time by indexing the corresponding gene sequence by name.

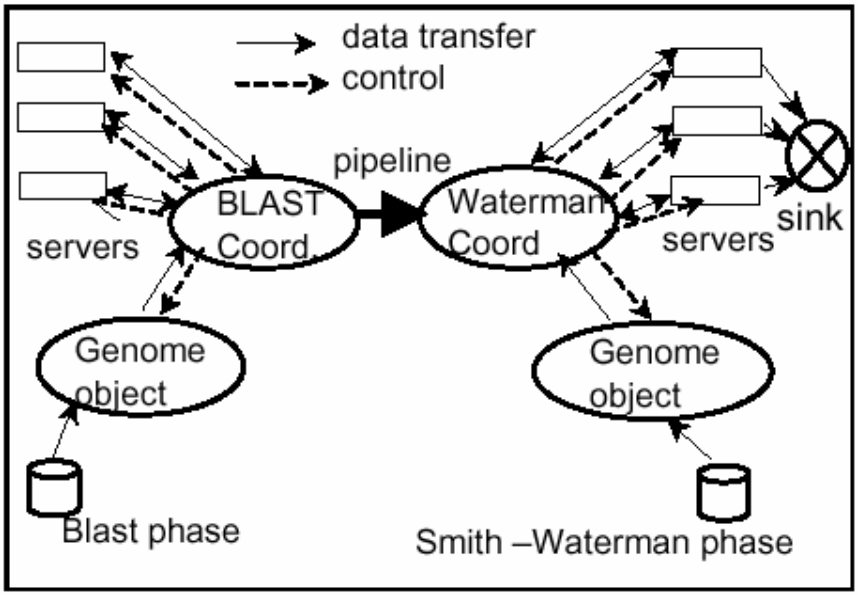



Figure 12. The overall distributed scheme

The advantages of this architecture:

1. It offers the linear speed up.
2. The system can harness thousands of inexpensive processors on the Internet and LANs in a scalable manner.

*Time complexity*: (**O(mn)**) **/ (M/c)** where **M** is number of genes in a genome and **c** is a constant.

# Chapter 7

# Implementation and experimental results

## 7.1 First Phase Algorithm

Welcome to My World!

Match a Patteren!

Play with Wild Cards!

Welcome To the World of Wild Cards!

Enter the Pattern with Wild Card (? / *): abc

Reset

**Matching Patteren "abc" with all the records of DataBase!**

Record no: 1 :gutyyutgj87

Pattern "abc" is not found!

Record no: 2 :675vdadxczd

Pattern "abc" is not found!

Record no: 3 :abfrabat

Pattern "abc" is not found!

Record no: 4 :cdabab

Pattern "abc" is not found!

Total pattern match in the Table are: 0

**Matching Patteren " ac t g " with all the records of DataBase!**

Record no: 1 :gutyyutgj87

Pattern " ac t g " is not found!

Record no: 2 :675vdadxczd

Pattern " ac t g " is not found!

Record no: 3 :abfrabat

Pattern " ac t g " is not found!

Record no: 4 :cdabab

Pattern " ac t g " is not found!

Total pattern match in the Table are: 0

### 7.2 Second Phase Algorithm

**Matching Patteren "ab*" with all the records of DataBase!**

Record no: 1 :gutyyutgj87

Pattern "ab*" is not found!

Record no: 2 :675vdadxczd

Pattern "ab*" is not found!

Record no: 3 :abfrabat

Patterns of type: ab* are abfrabat

Record no: 4 :cdabab

Patterns of type: ab* are abab

Total pattern match in the Table are: 2

**Matching Patteren "a t c g?" with all the records of DataBase!**

Record no: 1 :gutyyutgj87

Pattern "a t c g?" is not found!

Record no: 2 :675vdadxczd

Pattern "a t c g?" is not found!

Record no: 3 :abfrabat

Pattern "a t c g?" is not found!

Record no: 4 :cdabab

Pattern "a t c g?" is not found!

Total pattern match in the Table are: 0

**Results**

We evaluated the proposed two-phase searching algorithm denoted by 2-PSA. and we compared this algorithm with two sequential pattern mining algorithm developed in the data-mining field [1, 3, 5], which have shown superior performance on biosequences compared to earlier algorithms [6, 7]. Our purpose is to see how this algorithm would respond to the new type of explosion in biological sequences. All experiments have been executed on a PC with 2.40 GHZ CPU and 40 GB memory

running the Microsoft Windows 2000 Professional using the platform of Java development Kit ( JSP and Tomcat Server).

**Synthetic data Set**

Table : Parameters of the data used [7]

| **Symbol** | **Description** |
|---|---|
| **D** | Number of customers (= size of Database) =number of sequences |
| **C** | Average number of transactions per Customer =length of sequences |
| **T** | Average number of items per Transaction =1 |
| **S** | Average length of maximal potentially frequent sequences |
| **I** | Average size of Itemsets in maximal potentially frequent sequences =1 |
| **Ns** | Number of maximal potentially frequent Sequences |
| **Ni** | Number of maximal potentially frequent Itemsets = N |
| **N** | Number of items = 4 or 20 |

The first set of experiments was conducted on the synthetic data sets generated in [1,7]. Table 1 shows the parameters used. The alphabet size $N$ and sequence length $C$ characterizes the explosion of search space. The data sets with $N = 4$ simulate DNA sequences, the data sets with $N = 20$ simulate protein sequences, and the data set with $N = 10000$ simulates transaction sequences. The DNA or protein sequences have significantly longer average length C.

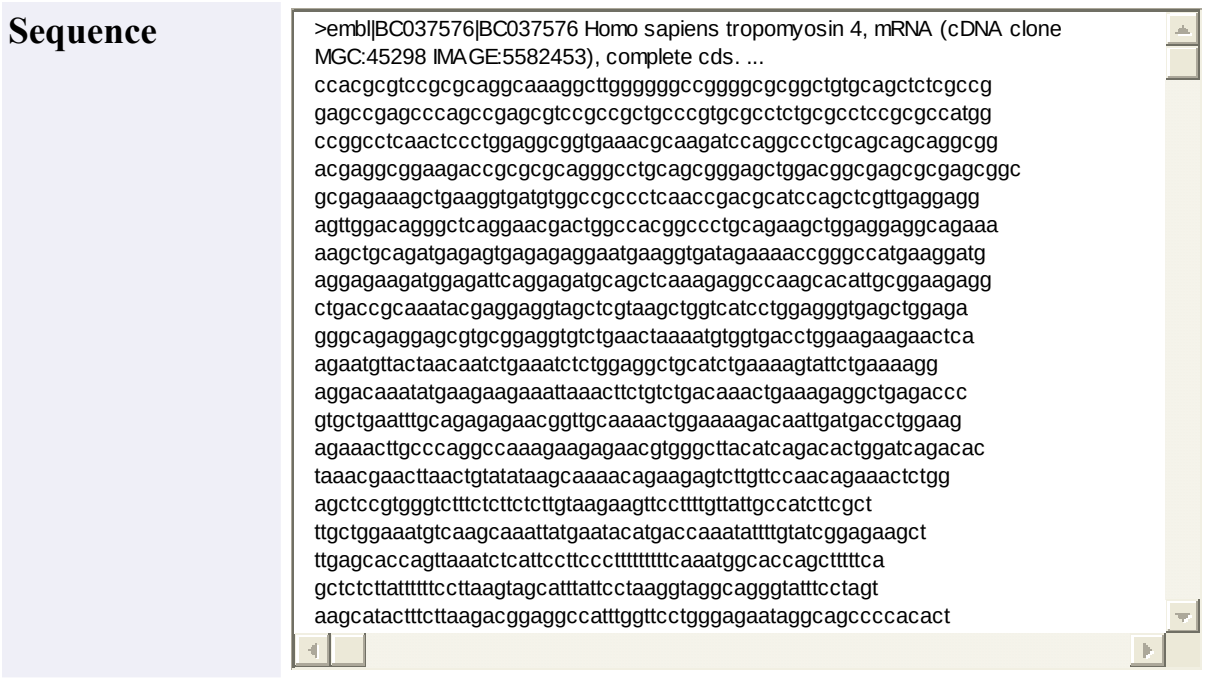
**Biological data sets**

The second set of experiments was conducted on real life mRNA sequences extracted from the web site of EMBL-EBI (European Bioinformatics Institute) .

MEDLINE; 91322517. PUBMED; 1907511. Oxtoby E., Dunn M.A., Pancoro A., Hughes M.A.;

"Nucleotide and derived amino acid sequence of the cyanogenic beta-glucosidase (linamarase) from white clover (Trifolium repens L.)."; Plant Mol. Biol. 17(2):209-219(1991). Submitted (19-NOV-1990) to the EMBL/GenBank/DDBJ databases. M.A. Hughes, UNIVERSITY OF NEWCASTLE UPON TYNE, MEDICAL SCHOOL, NEW CASTLE UPON TYNE, NE2 4HH, UKBottom of Form

| General Information | |
|---|---|
| Primary Accession # | BC037576 |
| Accession # | BC037576 |
| Entry Name | EMBL:BC037576 |
| Molecule Type | mRNA |
| Sequence Length | 1716 |
| Entry Division | HUM |
| Sequence Version | BC037576.1 |
| Creation Date | 18-SEP-2002 |
| Modification Date | 17-APR-2005 |
| **Description** | |
| Description | Homo sapiens tropomyosin 4, mRNA (cDNA clone MGC:45298 IMAGE:5582453), complete cds. |
| Keywords | MGC. ; |
| Organism | Homo sapiens (human) |
| Organism Classification | Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia; Eutheria; Euarchontoglires; Primates; Catarrhini; Hominidae; Homo. ; |
| **Sequence** | |
| Characteristics | Length: 1716 BP, A Count:489, C Count:386, G Count:468, T Count:373, Others Count:0 |




## 7.3 Sequence Database of DNA

ID BC037576_3; parent: BC037576
AC BC037576;
FT CDS 117..863
FT /codon_start=1
FT /db_xref="GOA:P07226"
FT /db_xref="GOA:P67936"
FT /db_xref="HSSP:1C1G"
FT /db_xref="UniProt/Swiss-Prot:P67936"
FT /gene="TPM4"
FT /product="tropomyosin 4"
FT /protein_id="AAH37576.1"

SQ Sequence 747 BP;

atggccggcc tcaactccct ggaggcggtg aaacgcaaga tccaggccct gcagcagcag 60
gcggacgagg cggaagaccg cgcgcagggc ctgcagcggg agctggacgg cgagcgcgag 120
cggcgcgaga aagctgaagg tgatgtggcc gccctcaacc gacgcatcca gctcgttgag 180

gaggagttgg acagggctca ggaacgactg gccacggccc tgcagaagct ggaggaggca 240
gaaaaagctg cagatgagag tgagagagga atgaaggtga tagaaaaccg ggccatgaag 300
gatgaggaga agatggagat tcaggagatg cagctcaaag aggccaagca cattgcggaa 360
gaggctgacc gcaaatacga ggaggtagct cgtaagctgg tcatcctgga gggtgagctg 420
gagagggcag aggagcgtgc ggaggtgtct gaactaaaat gtggtgacct ggaagaagaa 480
ctcaagaatg ttactaacaa tctgaaatct ctggaggctg catctgaaaa gtattctgaa 540
aaggaggaca aatatgaaga agaaattaaa cttctgtctg acaaactgaa agaggctgag 600
acccgtgctg aatttgcaga gagaacggtt gcaaaactgg aaaagacaat tgatgacctg 660
gaagagaaac ttgcccaggc caaagaagag aacgtgggct tacatcagac actggatcag 720
acactaaacg aacttaactg tatataa 747

## Execution time

Figures 1-4 shows the result on biosequences [1]. The execution time includes the time in segment phase that is computing frequent segments and pattern phase.

Figure 1: Graph between base sequences as input and the execution time

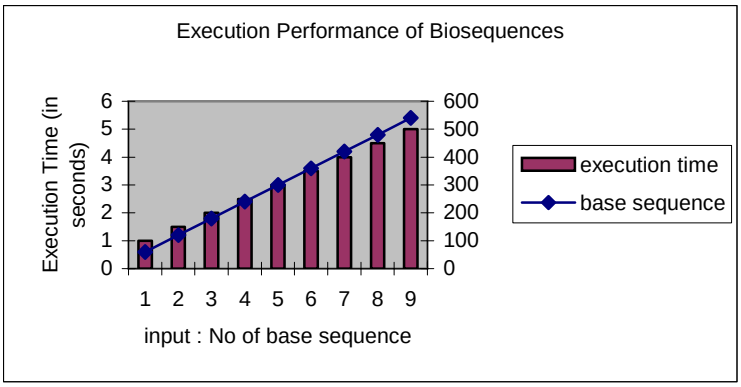



Figure 2: Graph for Scalability with respect to the database size

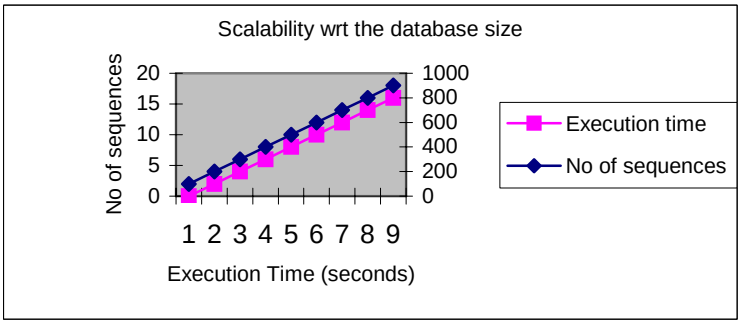


Figure 3: Graph for Scalability with respect to the database size

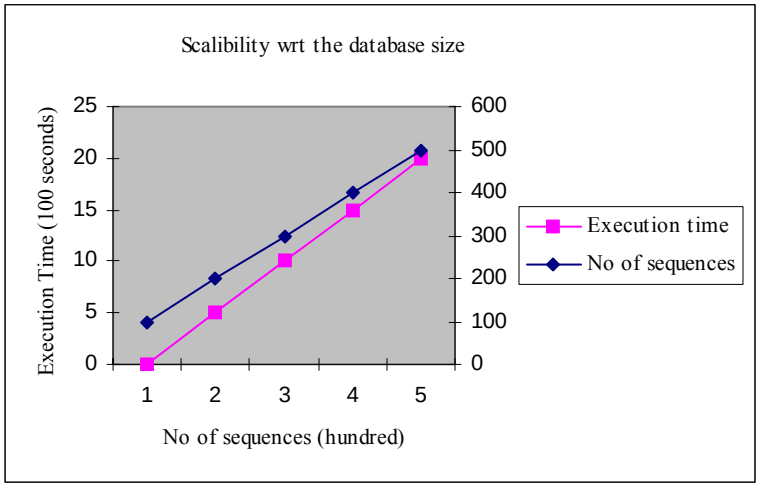
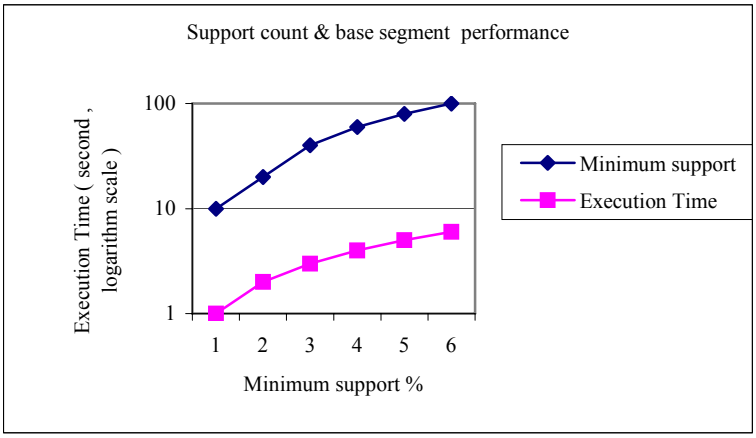



Figure 4: Graph between minimum support and the execution time



# Chapter 8

# EXTENSION & CONCLUSION

## 8.1 EXTENSION - Proposed Future Work

We have the following solution set in our mind as the target for future work.

- Distributed multi-database implementation.
- Web-based approach.
- Extension or integration with other bioinformatics applications (data mining, Image mining etc.).
- A framework for parallel Data Mining

## 8.2 CONCLUSION

Biosequences experience a different type of explosion of search space from the transaction point of view since traditional pruning techniques are not effective for mining biosequences. We proposed a two-phase searching algorithm 2-PSA to address this problem. The greatness is using the information obtained from the first phase is used in reducing the search in second phase. Results demonstrate significant improvement over the two-phase sequential pattern mining algorithm. Finally, we conclude the problems in this area of biological database searching and we can describe the problems as below:

- Currently practicing methods (BLAST and FASTA) becomes insufficient for rapidly growing databases.
- Newly proposed techniques look good on paper. But some are proposed only at the low level. All these techniques require thorough testing with live data. Almost all authors claims that his/her technique is the best. Nevertheless, each technique has its own beauty and virtue.
- Collaboration efforts for integrity and interpretability among the database are still needed.

Hence, it can be concluded that the technology in this area of research is not saturated yet. It is open for us to make new contributions.

Chapter 9

# Publicly Available Software Tools

***Sequence Similarity Search***

- BLAST -Basic local alignment search tool @ NCBI
- Blast Search @EBI
- Blast Search @Expasy
- Blast Search @ISREC
- Blast Search @Pasteur Institute , France

- PSI BLAST -Position Specific Iterated Blast search @ NCBI
- *C. elegans* Blast Server @ Sanger Centre: Sear
- FASTA Fasta or fastx search @ EBI
- BLAST Microbial Genomes -Genomic sequences search @ NCBI
- MPsrch -Smith-Waterman algorithm-based search
- BLASTPAT - BLAST-based Pattern Database Search
- FASTPAT - FASTA-based Pattern Database Search
- SectionSearch -FastA or TFastA search against predefined Sequence database
- WU-BLAST Archives - Washington University School of Medicine, St. Louis

# Databases of Patterns

### Sequence Database

- NCBI - National Center for Biotechnology Information (GenBank)
- EBI - European Bioinformatics Institute (EMBL)
- EMNEW - Index of New EMBL Sequences ( EBI)
- DDBJ - DNA Data Bank of Japan
- SWISS-PROT- Protein sequence database
- SWISSNEW - New SwissProt Sequence Entries @ EBI

- PIR - Protein Information Resource
- MIPS - Munich Information centre for Protein Sequences

| | |
|---|---|
| yeast | Yeast (Saccharomyces cerevisiae) genomic nucleotide sequences |
| pdb | Sequences derived from the 3-dimensional structure from Brookhaven Protein Data Bank |
| kabat [kabatnuc] | Kabat's database of sequences of immunological interest |
| vector | Vector subset of GenBank (R), NCBI, in ftp://ncbi.nlm.nih.gov/blast/db/ |
| mito | Database of mitochondrial sequences |
| Alu | Select Alu repeats from REPBASE, suitable for masking Alu repeats from query sequences |
| epd | Eukaryotic Promotor Database |

## 10. REFERENCES


http://www.ncbi.nlm.nih.gov [NCBI - National Center for Biotechnology Information (GenBank]

[1] Ke Wang, Yabo Xu, Jeffrey Xu Yu "Scalable Sequential Pattern Mining for Biological Sequences" Proceedings of the thirteenth ACM conference on Information and knowledge management November 2004

[2] R.C. Agarwal , C.C. Aggarwal, and V.V.V. Prasad, Depth first generation of long patterns, SIGKDD, 2000.

[3] J. Ayres , J.Gehrke, T. Yiu, and J. Flannick, Sequential pattern mining using a bitmap representation, SIGKDD, 2002,pp. 215 – 224.

[4] H.Mannila, H. Toivonen, and A. I. Verkamo, Discovery of frequent episodes in event sequences, Journal of Data Mining and Knowledge Discovery, Vol. 1, pp. 259-289, 1997.

[5] J. Pei, J. Han, B. Asl, Q. Chen, U. Dayal, and M. Hsu, Prefixspan: mining sequential patterns efficiently by prefix-projected pattern growth, ICDE, 2001.

[6] M. J. Zaki, SPADE: An efficient algorithm foe mining frequent sequences, Machine Learning Journal, Special Issue on Unsupervised Learning, Vol. 42, No. 1/2 pp. 31-60, 2001.

[7] R. Agrawal and R. Srikant, Mining Sequential Patterns, ICDE, 1995

[8] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad, Depth¯First generation of long patterns, SIGKDD, 2000.

[9] [RCSB, 2001] RSCB. Homepage of Protein DataBank. In Research Collaboratory for Structural Bioinformatics Website. 2001. http://www.rcsb.org/pdb/

[10.] [Luscombe et al., 2001] N. M. Luscombe, D. Greenbaum and M. Gerstein. What is Bioinformatics? An Introduction and Overview. In Yearbook of Medical Informatics. 2001. http://citeseer.nj.nec.com/421621.html

[11.] [Martins, 2000] W. S. Martins. Discovery Bioinformatics Lecture Notes. University of Delaware. 2000. http://www.capsl.udel.edu/courses/eleg667/2000/Slideindex.html

[12.] Brazma, A.; Jonassen, I.; Ukkonen, E.; and Vilo, J. 1996. Discovering patterns and subfamilies in biosequences. In States, D. J.; Agarwal, P.; Gaasterland, T.; Hunter, L.; and Smith, R., eds., Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology, 34–43. Menlo Park: AAAI Press. Brazma, A.; Vilo,

[13] D. Burdick, M. Calimlim, and J. Gehrke, MAFIA: A maximal frequent itemset algorithm for transactional databases, ICDE, 2001.

[14] Jones, N. and Pevzner, P. An Introduction to Bioinformatics Algorithms. MIT Press, Cambridge, MA, 2004.

[15] J. Yang, W. Wang, P.S. Yu, and J. Han, Mining long sequential patterns in a noisy environment, SIGMOD, 2002.

[16] Techniques for comparison, pattern matching and pattern discovery From sequences to protein topology David GILBERT1, David WESTHEAD2 and Juris VIKSNA3 (1) Bioinformatics Research Centre, Department of Computing Science University of Glasgow, Glasgow G12 8QQ, Scotland, UK (2) School of Biochemistry and Molecular Biology, University of Leeds, Leeds, West Yorkshire, LS2 9JT, U.K.(3) Institute of Mathematics and Computer Science, University of Latvia, Latvia 2001