

**A TOP-DOWN APPROACH FOR MINING MOST
SPECIFIC FREQUENT PATTERNS IN BIOLOGICAL
SEQUENCE DATA**

by

Xiang Zhang

B.Sc., Huazhong University of Science and Technology, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

© Xiang Zhang 2003
SIMON FRASER UNIVERSITY
September 2003

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without the permission of the author.

APPROVAL

Name: Xiang Zhang
Degree: Master of Science
Title of thesis: A TOP-DOWN APPROACH FOR MINING MOST SPECIFIC FREQUENT PATTERNS IN BIOLOGICAL SEQUENCE DATA

Examining Committee: Dr. Binay Bhattacharya
Chair

Dr. Martin Ester, Senior Supervisor

Dr. Ke Wang, Supervisor

Dr. S. Cenk Sahinalp, SFU Examiner

Date Approved:

October 29, 2003

SIMON FRASER UNIVERSITY

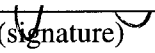
PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project and extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

**A Top-Down Method for Mining Most Specific
Frequent Patterns in Biological Sequence Data**

Author:


(signature)

Xiang Zhang

(name)

Nov. 26, 2003

(date)

Abstract

The emergence of automated high-throughput sequencing technologies has resulted in a huge increase of the amount of DNA and protein sequences available in public databases. A promising approach for mining such biological sequence data is mining frequent subsequences. One way to limit the number of patterns discovered is to determine only the most specific frequent subsequences which subsume a large number of more general patterns. In the biological domain, a wealth of knowledge on the relationships between the symbols of the underlying alphabets (in particular, amino-acids) of the sequences has been acquired, which can be represented in concept graphs. Using such concept graphs, much longer frequent patterns can be discovered which are more meaningful from a biological point of view. In this paper, we introduce the problem of mining most specific frequent patterns in biological data in the presence of concept graphs. While the well-known methods for frequent sequence mining typically follow the paradigm of bottom-up pattern generation, we present a novel top-down method (ToMMS) for mining such patterns. ToMMS (1) always generates more specific patterns before more general ones and (2) performs only minimal generalizations of infrequent candidate sequences. Due to these properties, the number of patterns generated and tested is minimized. Our experimental results demonstrate that ToMMS clearly outperforms state-of-the-art methods from the bioinformatics community as well as from the data mining community for reasonably low minimum support thresholds.

Acknowledgments

I would like to express my deepest gratitude to my senior supervisor, Dr. Martin Ester. He has provided me with inspiration both professionally and personally during the course of my degree. I am thankful to him for sharing his ideas and experience, for providing insightful directions for work and for his flexibility and patience.

I am very grateful to Dr. Ke Wang for being my supervisory committee member and Dr. Cenk Sahinalp for serving as examiner of this thesis. They were generous with their time to read this thesis carefully and make thoughtful suggestions.

Many thanks go to Gabor Melli for proof reading of my thesis.

I will also like to thank Carol, Kersti and Wilma for helping me with various administration stuffs.

Finally, I will thank my parents for their unconditional love and support.

Contents

Approval	ii
Abstract	iii
Acknowledgments	iv
Contents	v
List of Tables	vii
List of Figures	viii
1 Introduction	1
1.1 Motivation	1
1.2 Contribution	2
1.3 Organization of the Thesis	3
2 Background Information	4
2.1 Cells	4
2.2 Proteins	5
2.3 Genetic Sequences	9
2.3.1 DNA	10
2.3.2 RNA	13
3 Related Work	15
3.1 Pattern Enumeration Methods	15
3.1.1 GSP Algorithm	16

3.1.2	TEIRESIAS	17
3.1.3	Pattern Growth Method	19
3.2	Efficient Support Counting	20
3.2.1	PSP	21
3.2.2	SPADE	21
3.2.3	SPAM	22
3.3	Version Space	23
4	The Algorithm	28
4.1	Terminologies and Problem Definition	28
4.2	ToMMS	32
4.3	Concrete Patterns: Example and Analysis	34
4.4	Incorporating Concept Graphs	35
4.5	Determining the Maximum Length	41
4.6	Correctness of ToMMS	42
4.7	Efficient Implementation of the Match Operation	43
5	Experimental Evaluation	46
5.1	Datasets	46
5.2	Design of the Experiments	47
5.3	Experimental Results	48
6	Conclusion	53
6.1	Summary of Thesis	53
6.2	Future Research Directions	54
	Bibliography	56

List of Tables

2.1	The 20 amino acids along with their three and one letter codes.	7
2.2	The genetic code mapping codons to amino acids.	12
4.1	Notations	32
4.2	Example of function Generalize: length 4	39
4.3	Example of function Generalize: length 3	40
4.4	Example of function Generalize: length 2	40
5.1	Description of the evaluation datasets.	47
5.2	Concepts of amino acids and corresponding physio-chemical properties. . . .	47

List of Figures

2.1	Examples of amino acids: alanine and threonine.	6
2.2	Amino acids categorized according to the chemical nature of the R-group. . .	8
2.3	A polypeptide chain.	9
2.4	Primary, secondary, tertiary, and quaternary structures of proteins.	10
2.5	A schematic molecular structure view of one DNA strand.	11
2.6	A schematic molecular structure view of a double strand of DNA.	11
2.7	Genetic information flow in a cell.	13
3.1	The GSP Algorithm	16
3.2	ExtendPattern	18
3.3	Convolution phase of TEIRESIAS	19
3.4	Prefix tree vs. Hash tree	21
3.5	Horizontal vs Vertical Representation of Database	22
3.6	Horizontal vs Bitmap Representation of Database	23
3.7	Generation of a new bitmap	23
3.8	Candidate-Elimination Algorithm	26
4.1	Example of a concept graph	29
4.2	The ToMMS Algorithm	33
4.3	Dataset of running example	34
4.4	Example of search space of concrete pattern	35
4.5	Function Generalize	38
4.6	Example of modified prefix tree	44
4.7	Function Generalize	45
5.1	Runtime on outer membrane protein sequence dataset.	48

5.2	Runtime on non outer membrane protein sequence dataset.	49
5.3	Runtime on yeast protein sequence dataset.	50
5.4	Runtime of finding maximal length compare to runtime of ToMMS.	51
5.5	Scalability of ToMMS.	52
5.6	Number of patterns discovered by ToMMS, Teiresias and SPAM.	52

Chapter 1

Introduction

1.1 Motivation

The emergence of automated high-throughput sequencing technologies has resulted in a significant increase of the amount of DNA and protein sequences available in public databases such as GenBank [4] and SWISS-PROT [5]. To make sense of this flood of data, molecular biologists now try to uncover the function of different genes and proteins in the corresponding biological systems. For example, they want to identify genes in the DNA sequences or assign a protein sequence to one of the well-known protein families. Manual analysis of the large databases is infeasible, and data mining solves some of the challenge. A good number of methods for classification of sequences, finding frequent sequences or finding motifs have been presented in the literature.

One promising approach for mining biological sequence data is mining frequent patterns, i.e. patterns which occur in at least as many sequences as specified by some threshold (minimum support). This approach is motivated by two fundamental biological facts: (1) similar sequences have the same or similar function with a high probability and (2) typically large portions of DNA or protein sequences are considered to be noise. Thus, the sequential patterns determining the function are expected to be relatively short (compared to the sequence length) and to occur much more frequently than (random) noise patterns.

In the KDD community, methods for mining frequent sequences have received a lot of attention (GSP [25], PrefixSpan [20], SPADE [30], SPAM [3]). All these methods belong to the class of bottom-up pattern enumeration methods, i.e. they start with short frequent patterns and continue to extend them until they become infrequent. Unfortunately, most

of these methods suffer from two major weaknesses which seriously limit their usefulness in the biological domain:

- These methods generate all frequent patterns which typically leads to very large numbers of patterns since each subpattern of a frequent pattern is also frequent.
- The frequent patterns are typically too short to be meaningful when using only the original alphabet for the patterns.

To overcome these limitations, we present a new method for frequent sequence mining with the following properties:

- We restrict the result of pattern mining to the subset of most specific frequent patterns, i.e. frequent patterns for which no specialization is still frequent. All other frequent patterns can be derived from these patterns on demand.
- Often some symbol can be replaced by a similar one without loss of function of the whole pattern. In particular for proteins, a wealth of knowledge on the relationships between different amino-acids has been acquired, which can be represented in concept graphs. Using such concept graphs, much longer frequent patterns can be discovered which are more meaningful from a biological point of view.

In the bioinformatics community, the Teiresias algorithm [21] has been proposed which takes a similar approach. It determines the set of all closed frequent patterns, which is the set of all frequent patterns for which all extensions have a smaller support. Teiresias also uses concept graphs to discover generalized patterns. Like the methods from the KDD community, Teiresias adopts the paradigm of bottom-up pattern enumeration.

1.2 Contribution

In this thesis, we introduce a novel approach for mining most specific frequent patterns performing top-down pattern enumeration. The notion of most specific frequent patterns generalizes the notion of maximal frequent patterns in the presence of concept graphs. A frequent pattern is most specific if no specialization of this pattern is frequent, where a specialization can either be an extension of the pattern by one element or a replacement of one element of the pattern by a predecessor from the concept graph. Top-down pattern

enumeration starts with infrequent patterns of maximal length and performs generalizations (the inverse operation of a specialization) until the patterns become frequent. This approach is more appropriate than the classic bottom-up approach for mining most specific (not all) patterns if these patterns are long and if we can reliably estimate the maximal length of frequent patterns. We will show later how to efficiently determine even the exact value of this maximal length. Our experimental evaluation on real life biological sequence datasets will demonstrate that the proposed top-down method clearly outperforms the state-of-the-art methods from the KDD as well as from the bioinformatics community for low minimum support values. Low minimum support values are necessary to discover frequent patterns long enough to be biologically meaningful.

ToMMS has the following advantages that existing algorithms do not have. First, ToMMS is a pure top-down approach. Although some existing algorithms employ mechanisms such as look-ahead to discover long patterns first, their basic approach is still using short patterns to generate longer patterns, i.e. bottom-up search. Second, ToMMS will never enumerate a potential pattern which does not exist in the database, i.e. the enumerated patterns at least have one supporting sequence. This will greatly reduce the effort of exploring the search space. Third, ToMMS can efficiently determine the maximal length of all frequent patterns before enumerating all of them.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 3 surveys related work. In chapter 4, we introduce our notion of most specific frequent patterns and present ToMMS, a new top-down method for mining such patterns. In chapter 5, we report the results of an experimental evaluation and comparison with state-of-the-art methods. Chapter 6 summarizes our contributions and outlines directions for future research.

Chapter 2

Background Information

This section provides readers with enough information so that they can comfortably follow the biological background of this thesis. The characteristics of cells, proteins, DNA and RNA will be described in detail. There are excellent books [12] [28] that provide a full fledged introduction to Molecular Biology.

2.1 Cells

Life is the subject of Biology. In nature we find both living and non-living things. Yet research in the past centuries reveals that both kinds of matter are composed by the same atoms and conform to the same physical and chemical rules. What distinguishes living from non-living chemical forms? Life can be characterized by its properties. Living things can move, reproduce, grow, eat, and so on. They have an active participation in their environment, as opposed to non-living things. A system is thought to be “living” if it has following three general characteristics:

- metabolism
- growth
- reproduction

The most elementary unit exhibiting these properties is the cell. Cells come in two basic varieties. *Prokaryotic* cells are the simplest form, composed of a single compartment which is protected from the outside with a membrane called the cell wall. *Eucaryotic* cells

have a central, all encompassing compartment which contains many smaller compartments. The most important among these compartments is the *nucleus*, which contains the genetic material of the eucaryotic cells. Procaryotic cell are found only in bacteria while higher organisms are composed of one or more eucaryotic cells.

About 90% of the cell is water. The remaining 10% contains two types of molecules:

- Elementary molecules: these are small molecules created by a variety of chemical reactions in the cell. Most important among them are the *nucleotides* and the *amino acids*. Elementary molecules provide the building material for polymeric molecules.
- Polymeric molecules: these are the structural components of the cell. They are formed when elementary molecules bind together into long chains. The two important polymeric molecules are (1) the *nucleic acids* (DNA, RNA) and (2) the *proteins*. Nucleic acids are polymeric assemblies of nucleotides while the proteins are chains of amino acids. Nucleic acids and proteins are sometimes collectively referred to as *macromolecules* or *biosequences* or *biological sequences*.

2.2 Proteins

Proteins are the most important macromolecules. They are responsible for almost the entire repertoire of biochemical reactions taking place inside the cell. Most substances in our bodies are proteins, of which there are many different kinds. Some of them include:

- Structural proteins: they are the building blocks of the various tissues.
- Enzymes: they catalyze vital chemical reactions that would otherwise take too long to complete
- Transporters: they carry chemical elements from one part of the organism to another.
- Antibody proteins: they are part of the immune system.

Proteins are chains of simpler molecules called amino acids. There are 20 different amino acids. Examples of amino acids can be seen in Figure 2.1. Every amino acid has one central carbon atom to which four chemical groups attached via covalent bonds: a hydrogen atom(H), an amino group (NH_2), a carboxyl group (COOH) and a side chain (R). It is the side chain that distinguishes one amino acid from another. Side chains can be as simple as

one hydrogen atom or as complicated as two carbon rings. Table 2.1 lists all the 20 amino acids along with the three-letter and one-letter codes used for each.

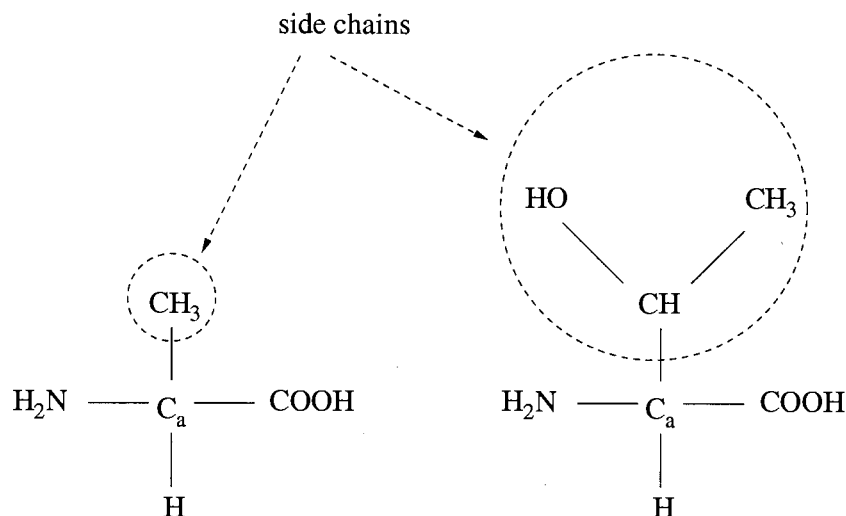


Figure 2.1: Examples of amino acids: alanine and threonine.

Amino acids can be partitioned into different sets by their physio-chemical properties [26] [24] or empirical results got from experiments performed on protein databases [11] [29]. Figure 2.2 shows a well known hierarchical group where amino acids are categorized according to the chemical nature of the R-group. Amino acids belonging to the same class are said to be chemically similar. Please note that the relationship is not necessarily tree structure. Sometime it can be graph.

In a protein, amino acids are joined by *peptide bonds*. For this reason, proteins are polypeptides chains. Binding occurs when the carboxyl group of an amino acid interacts with the amino group of the next amino acid. The result of this process is (1) the formation of a bond between the two amino acids and (2) the generation of a water molecule from atoms released from the carboxyl group and the amino group. Hence, what we really find inside a polypeptide chain is a *residue* of the original amino acid. Thus we generally speak of a protein having 100 residues, rather than 100 amino acids. Typical proteins contain about 300 residues, but there are proteins with as few as 100 or with as many as 5,000 residues.

The peptide bond makes every protein have a *backbone*, given by repetitions of the basic block $-N-C_\alpha-(CO)-$. To every C_α there corresponds a side chain. See figure 2.3 for a

Name	One-letter code	Three-letter code
Alanine	A	Ala
Cysteine	C	Cys
Aspartic Acid	D	Asp
Glutamic Acid	E	Glu
Phenylalanine	F	Phe
Glycine	G	Gly
Histidine	H	His
Isoleucine	I	Ile
Lysine	K	Lys
Leucine	L	Leu
Methionine	M	Met
Asparagine	N	Asn
Proline	P	Pro
Glutamine	Q	Gln
Arginine	R	Arg
Serine	S	Ser
Threonine	T	Thr
Valine	V	Val
Tryptophan	W	Trp
Tyrosine	Y	Tyr

Table 2.1: The 20 amino acids along with their three and one letter codes.

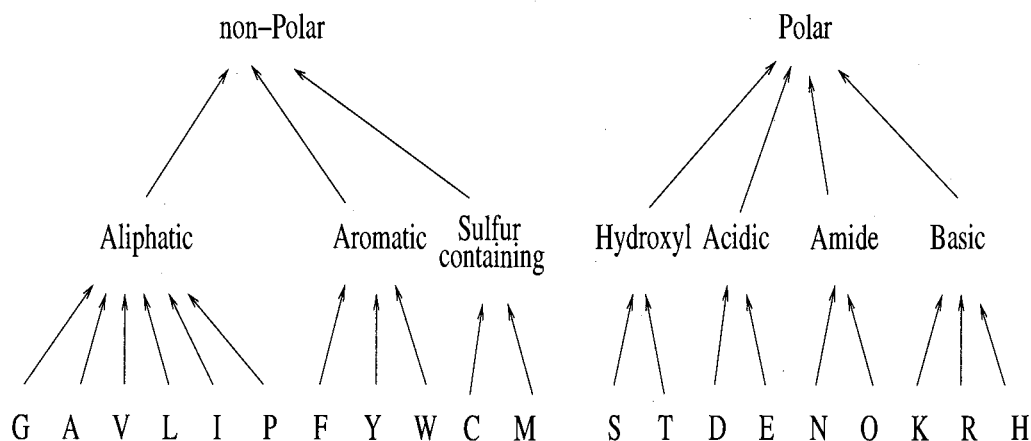


Figure 2.2: Amino acids categorized according to the chemical nature of the R-group.

schematic view of a polypeptide chain. Because we have an amino group at one end of the backbone and a carboxyl group at the other end, it is possible to use these two groups in order to impose a notion of direction on a protein. The convention is that polypeptides begin at the amino group (*N-terminal*) and end at the carboxyl group (*C-terminal*).

Up to now, we have been thinking that a protein is a linear sequences of residues. This will be the norm throughout this thesis. The representation of a protein as a string of its constituent amino acids is known as its *1-dimensional* or *primary structure*. Proteins actually fold in three dimensions, presenting *secondary*, *tertiary*, and *quaternary* structures. The fold of a protein is of extreme importance because it typically provides clues about the function of the protein. A protein's secondary structure is formed through interactions between backbone atoms only and results in local structures such as helices. Tertiary structures are the result of secondary structure packing on a more global level. Yet another level of packing, or a group of different proteins packed together, receives the name of quaternary structure. Figure 2.4 depicts these structures schematically.

Proteins can fold in three dimensions because the plane of the bond between the C_α atom and the nitrogen atom may rotate, as can the plan between the C_α atom and the other C atom. These rotation angles are known as ϕ and ψ , respectively, and are illustrated in figure 2.3. Side chains can also move, but it is a secondary movement with respect to the backbone rotation. Thus if we specify the values of all $\phi - \psi$ pairs in a protein, we know its exact folding. Determining the folding, or three-dimensional structure, of a protein is one of the main research areas in molecular biology, for three reasons. First,

the three-dimensional shape of a protein is related to its function. Second, the fact that a protein can be made out of 20 different kinds of amino acids makes the resulting three-dimensional structure in many cases very complex and without symmetry. Third, no simple and accurate method for determining the three-dimensional structure is known. Existing techniques for solving protein structures, such as X-ray crystallography and NMR, are very time consuming. Furthermore, there are many proteins which are not amenable to study by either of these two techniques.

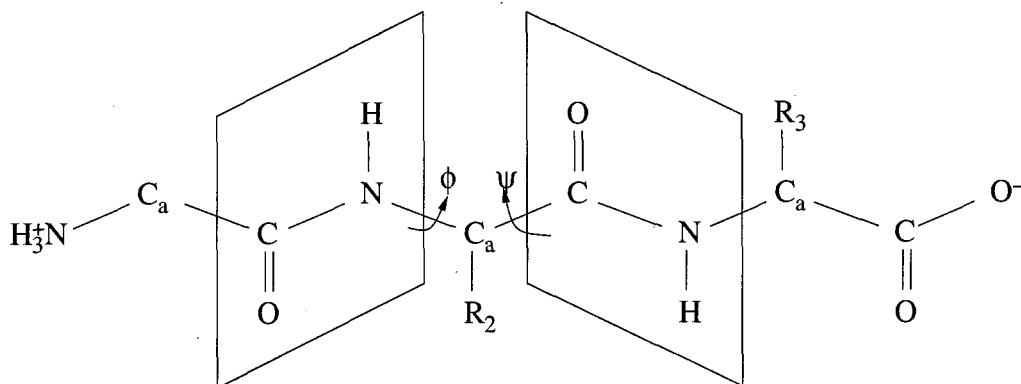


Figure 2.3: A polypeptide chain.

The three-dimensional shape of a protein determines its function in the following way. A folded protein has an irregular shape. This means that it has varied nooks and bulges, and such shapes enable the protein to come in closer contact with, or bind to, some other specific molecules a protein can bind to depend on its shape. For example, the shape of a protein can be such that it is able to bind with several identical copies of itself, building, say, a thread of hair. Or the shape can be such that molecules A and B bind to the protein and thereby start exchanging atoms. In other words, a reaction takes place between A and B, and the protein is fulfilling its role as a catalyst.

2.3 Genetic Sequences

Living organisms contain two kinds of nucleic acids: deoxyribonucleic acid, better known as DNA, and ribonucleic acid, or RNA.

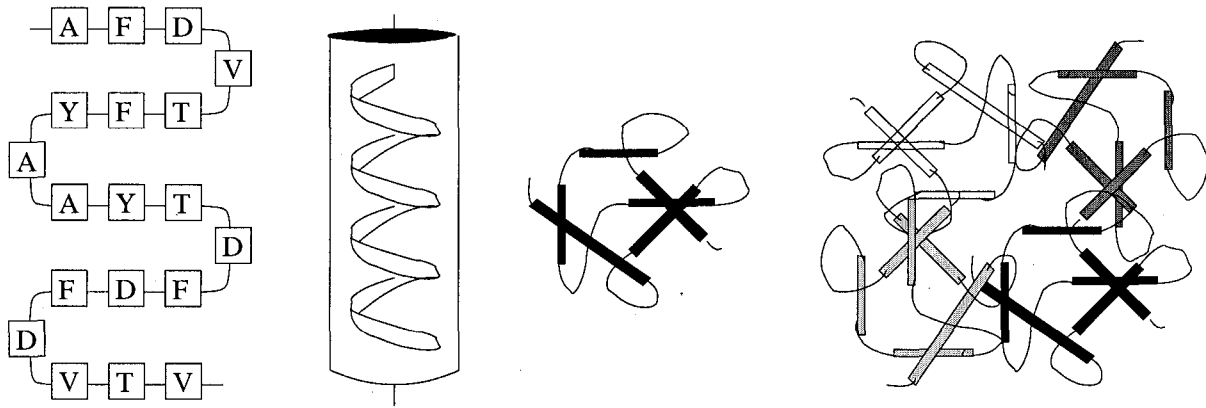


Figure 2.4: Primary, secondary, tertiary, and quaternary structures of proteins.

2.3.1 DNA

Molecules of DNA are chains of nucleotides. A single DNA molecule can be millions of *bases* long (nucleotides are alternatively called base). There are four different bases, i.e. adenine (A), guanine (G), cytosine (C) and thymine (T).

The mechanics of DNA formation are similar to that of proteins. Nucleotides have a basic part which is common to all of them. The side group attached to the basic part distinguishes one nucleotide from the other. Long chains are formed when series of nucleotides bind together through *sugar-phosphate* bonds. These bonds hold together the basic parts of successive nucleotides, in the same way that peptide bonds hold together the amino acids of a protein, which is shown in figure 2.5. The assembly of the basic parts in a DNA molecule is referred to as the *backbone*. DNA molecules are double stands. The two strands are tied together in a helical structure. Each base in one strand is paired with a base in the other strand. Base A is always paired with base T, and C is always paired with G. Because of these rules, either of the two strands uniquely defines the other and can be used to describe the corresponding DNA molecule. Figure 2.6 presents a schematic molecular structure view of a double strand of DNA. The two strands are connected through hydrogen bonds between the side groups of facing nucleotides. As a result, a DNA molecule can be represented as a single string over the alphabet of the nucleotides.

The entire DNA of an organism comprises that organism's *genome*. The size of a genome depends on the complexity of the host organism. The human genome, for example, has an estimated 3×10^9 *base pairs* (a base pair is another way to refer to a pair of facing nucleotides

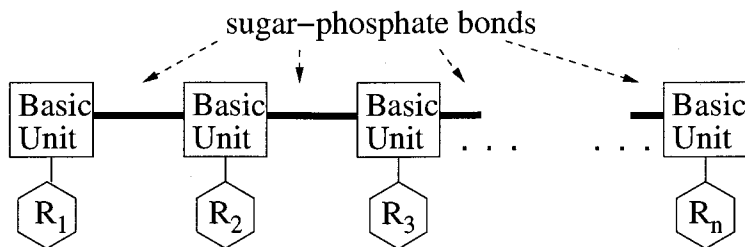


Figure 2.5: A schematic molecular structure view of one DNA strand.

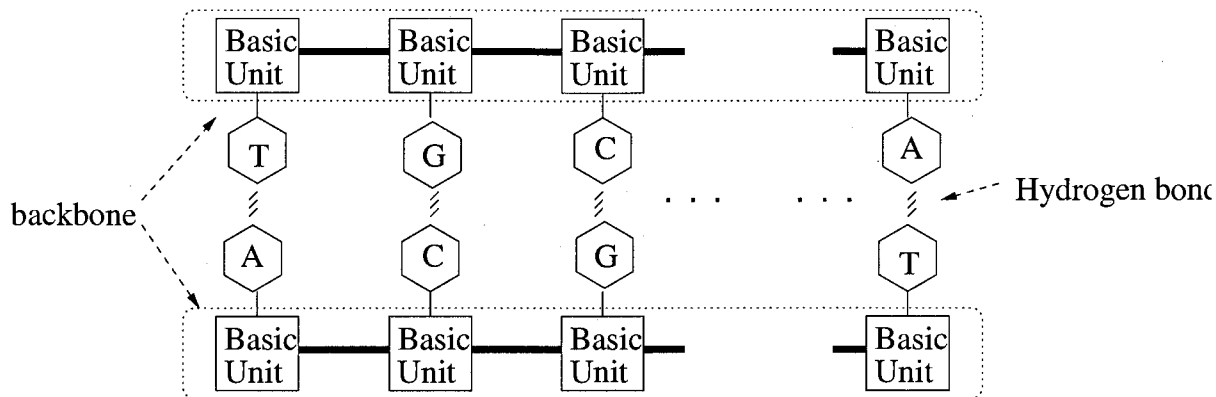


Figure 2.6: A schematic molecular structure view of a double strand of DNA.

within the DNA double helix.)

DNA is genetic material. It carries hereditary information from parents to children. DNA encode information for building proteins. DNA is broken into triplets and every triplet is translated into an amino acid. Each nucleotide triplet is called a *codon*. The mapping between triplets of DNA bases and amino acids is called the *genetic code*. The genetic code is shown in Table 2.2. Notice that there are 64 possible nucleotide triplets, but there are only 20 amino acids to specify. The consequence is that different triplets correspond to the same amino acid. There are three codons do not code for any amino acid and are used to signal the end of a protein coding region on a large DNA molecule. It is interesting to note that based on the genetic code, each concept graph on amino acids also implies a corresponding concept graph on codons.

Not all of the genome is used for coding proteins. Only certain contiguous stretches along DNA encode the information for building proteins. This coding parts are organized in *genes*, i.e. distinct regions of consecutive bases. Every gene codes for one particular protein.

First position	Second position				Third position
	G	A	C	U	
G	Gly	Glu	Ala	Val	G
	Gly	Glu	Ala	Val	A
	Gly	Asp	Ala	Val	C
	Gly	Asp	Ala	Val	U
A	Arg	Lys	Thr	Met	G
	Arg	Lys	Thr	Ile	A
	Ser	Asn	Thr	Ile	C
	Ser	Asn	Thr	Ile	U
C	Arg	Gln	Pro	Leu	G
	Arg	Gln	Pro	Leu	A
	Arg	His	Pro	Leu	C
	Gly	His	Pro	Leu	U
U	Trp	STOP	Ser	Leu	G
	STOP	STOP	Ser	Leu	A
	Cys	Tyr	Ser	Phe	C
	Cys	Tyr	Ser	Phe	U

Table 2.2: The genetic code mapping codons to amino acids.

Genes are flanked by *control regions* which mark the beginning and the end of these genes. The remaining DNA seems to be non-operational and is known as *junk DNA*. Interesting enough, the largest part of the genome in higher organisms is composed of junk DNA. In humans for example, about 95% of the DNA is non-coding.

2.3.2 RNA

RNA molecules are much like DNA molecules, with the following basic compositional and structural differences:

- RNA uses the nucleotide Uracil (U) where DNA would have used Thymine(T).
- The backbone-forming basic unit of the RNA nucleotides.
- RNA is single stranded. Sometimes we see RNA-DNA hybrid helices; also, parts of an RNA molecule may bind to other parts of the same molecule by complementarity. The three-dimensional structure of RNA is far more varied than that of DNA.

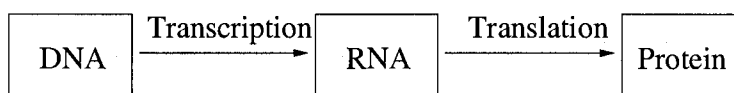


Figure 2.7: Genetic information flow in a cell.

How does the information in the DNA result in proteins? A cell mechanism recognizes the beginning of a gene thanks to a *promoter*. The promoter is a region before each gene in the DNA that serves as an indication to the cellular mechanism that a gene is ahead. Having recognized the beginning of a gene, a copy of the gene is made on an RNA molecule. This resulting RNA is the messenger RNA, or mRNA for short, and will have exactly the same sequence as one of the strands of the gene but substituting U for T. This process is called *transcription*. The mRNA will then be used in cellular structures called ribosomes to manufacture a protein. After transcription phase, another phase called *translation* will read the mRNA and create the final protein. The translation is accomplished by the ribosomes, complex bodies built from proteins and RNA (the ribosomal RNA is referred to as rRNA). Ribosomes are the chemical factories of the cell. They operate by walking over an mRNA molecule, reading the codons on the mRNA one by one, getting the appropriate amino acid for each codon and binding these amino acids together into the final protein chain. Transfer

RNAs, or tRNA are the molecules that actually implement the genetic code. They make the connection between a codon and the specific amino acid this codon codes for. Each tRNA molecule has, on one side, a conformation that has high affinity for a specific codon and, on the other side, a conformation that binds easily to the corresponding amino acid. As the mRNA passes through the interior of the ribosome, a tRNA matching the current codon, i.e. the codon in the mRNA currently inside the ribosome, binds to it, bringing along the corresponding amino acid (a generous supply of amino acids is always floating around in the cell). The three-dimensional position of all these molecules in this moment is such that, as the tRNA binds to its codon, its attached amino acid falls in place just next to the previous amino acid in the protein chain being formed. A suitable enzyme then catalyzes the addition of this current amino acid to the protein chain, releasing it from the tRNA. A protein is constructed residue by residue in this fashion. When a STOP codon appears, no tRNA associates with it, and the synthesis ends. The mRNA is released and degraded by cell mechanisms into ribonucleotide, which will be then recycled to make other RNA. Figure 2.7 summarizes the process just described.

Chapter 3

Related Work

The topic of mining frequent sequential patterns has received significant attention in the KDD community [2] [10] [27] [20]. In the bioinformatics community, a lot of research on mining patterns in biological sequence data has been conducted [24] [23] [21]. In this section, we survey important methods from both of these areas. Ignoring differences in definition of pattern language, existing sequential pattern discovery algorithm can be categorized into two categories: algorithms that concentrate on pruning search space and algorithms using special data structure for fast support counting. The following criteria will be used through the survey:

- In which direction is the pattern search space enumerated: Bottom-up or top-down?
- In which order is the pattern space searched: Depth-First or Breadth-First?
- Which patterns are reported: Maximal, closed or all patterns?
- Can the method handle concept graphs, i.e. does it find also generalized patterns using a given concept graph?

3.1 Pattern Enumeration Methods

Sequence pattern discovery problems are provably hard [13]. For example if we want to find amino acids pattern of length 10, there are potentially 20^{10} such patterns. Much research has been done on how to efficiently explore the huge search space. Many elegant

pruning techniques are proposed that make the search feasible for typical input data. In this subsection, we survey the algorithms that focus on these techniques.

3.1.1 GSP Algorithm

The sequential pattern mining problem was first introduced in KDD community by Agrawal and Srikant in [2]: *Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, and given a user specified minimum support threshold, sequential pattern mining is to find all of the frequent subsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than minimum support.* In [2] three algorithms are presented. The algorithm AprioriAll shown to perform better to the other two algorithms. The same authors in a later work [25] presented the GSP algorithm that outperforms AprioriAll by up to 20 times.

GSP is based on the *apriori* heuristic proposed in association mining [1]. The heuristic states the fact that any superpattern of a non frequent pattern cannot be frequent. The GSP algorithm is shown in figure 3.1.

Algorithm 3.1 Algorithm GSP

Input: a sequence database D and minimum support value δ

Output: the set of all frequent patterns

Method:

1. Function GSP(D, δ)
2. $F_1 = \{\text{frequent 1-sequences}\}$
3. $F = F_1$; /*set of all frequent patterns*/
4. **for** ($k=2$; $F_{k-1} \neq \emptyset$; $k++$) **do**
5. $C_k = \text{set of candidate k-sequences}$;
6. **for** all sequences s in database D **do**
7. increment count of all unique $\alpha \in C_k$ | α is a subsequence of s;
8. $F_k = \{\alpha \in C_k | \alpha \text{ is frequent}\}$;
9. $F = F \cup F_k$;
10. **return** F;

Figure 3.1: The GSP Algorithm

GSP adopts a bottom-up, breadth first search strategy, outlined as follows. The first database scan finds all of the frequent items which form the set of single item frequent sequences. Each subsequent pass starts with a seed set of sequential patterns, which is the set of sequential patterns found in the previous pass. This seed set is used to generate new potential patterns, called candidate sequences. Each candidate sequence contains one more

item than a seed sequential pattern, where each element in the pattern may contain one more item than a seed sequential pattern, where each element in the pattern may contain one item or multiple items. All the candidate sequences in a pass will have the same length. The scan of the database in one pass finds the support for each candidate sequences. All the candidates whose support in the database is no less than minimum support form the set of the newly found sequential patterns. This set then becomes the seed set for the next pass. The algorithm terminates when no new sequential pattern is found in a pass, or when no candidate sequence can be generated.

In candidate generation step, for any given pair of frequent length $k - 1$ sequence (p, p') , where discarding the first item of p and last item of p' results in identical patterns, create a new candidate pattern of length k by appending the last element of p' to p . Hence, it uses the *apriori* heuristic. For example, given frequent length 2 patterns AB and BC , they can be used to create a new candidate sequence ABC .

In GSP concept graphs are incorporated by simply transforming a sequence when counting its support.

3.1.2 TEIRESIAS

In the bioinformatics community, a wealth of methods for determining frequent sequential patterns has been developed [24] [23] [21]. [7] presents a good overview on this research. TEIRESIAS [21] can be considered as state-of-the-art method for mining all closed (called maximal in their paper) frequent patterns in a database. Closed patterns are patterns for which all extensions have a smaller support, and the set of closed patterns is typically a significantly larger superset of the set of all maximal patterns. We will next follow the author's definition and call the patterns reported by TEIRESIAS maximal patterns. While the original proposal did not involve concept graphs, it has later been extended to support this functionality.

TEIRESIAS searches for patterns consisting of characters of the alphabet Σ and wild-card characters $'.'$. Moreover, the patterns must satisfy certain density constraint, limiting the number of wild-cards occurring in any stretch of pattern, that is any subpattern of p containing exactly l non-wildcard characters has length at most l . Such patterns are called $\langle l, w \rangle$ patterns, where l and w are specified by user.

The basic idea of TEIRESIAS algorithm is that if a pattern p is a $\langle l, w \rangle$ pattern occurring in at least k sequences, then its subpatterns are also $\langle l, w \rangle$ patterns occurring at

least k sequences. Therefore the algorithm assembles the patterns from smaller subpatterns.

TEIRESIAS works in two phases. In the first phase, called scanning phase, it finds all $\langle l, w \rangle$ patterns occurring in at least k sequences that contain exactly l non-wildcards. In the second phase, convolution phase, these elementary patterns are extended by combining them together. The basic operation is to take two patterns p and q created so far, take the suffix of p containing exactly $l - 1$ non-wildcards, take prefix of q containing exactly $l - 1$ non-wildcards. If the suffix and the prefix are equal, p and q can be combined together so that the $l - 1$ non-wildcards overlap. If the resulting pattern occurs at least k times, we keep it, otherwise we discard it. For example let $p = AB.CD.E$ and $q = DFE.G$. Then p and q cannot be combined together, because $D.E \neq DF$. However if $q = D.E.G$, they can be combined together obtaining $AB.CD.E.G$.

The efficiency of TEIRESIAS is primarily due to a special join operation (convolution) which allows the extension of a pattern by more than one element at a time. The convolution phase of the TEIRESIAS algorithm can be described as follows: for each elementary pattern p (starting with the largest pattern in prefix ordering), try to extend pattern p with other elementary patterns. The figure below shows the procedure of how to extend pattern p .

Algorithm 3.2 Extend an element pattern (In convolution phase of Teiresias)

Input: an elementary pattern p and the set of all elementary patterns EP

Output: a potential closed pattern

Method:

1. Function ExtendPattern(p , EP)
2. While there exists a pattern $q \in EP$, which can be combined to the left side of p
3. Take such q which is largest in suffix ordering;
4. Let r be the pattern resulting from combining q to the left of p ;
5. If pattern r has number of occurrences at least k and is maximal so far
6. Try to extend pattern r with other elementary patterns;
- /*using the procedure "ExtendPattern" recursively*/
7. If pattern r has the same number of occurrences as pattern p , then pattern p is not maximal and we do not need to search for other extensions of p ;
- /*exit the procedure*/
8. otherwise pattern r is not significant pattern;
9. Repeat the same process for the elementary patterns which can be combined to the right side of p (starting with the largest pattern in prefix ordering);
10. Report pattern p ;

Figure 3.2: ExtendPattern

In the convolution phase we produce all possible patterns in this way. We take each

elementary pattern, and we try to extend it on both sides by gluing it with other elementary patterns in all possible ways(depth first search). The figure below shows an example of extending elementary pattern *F.AS*. Any pattern that cannot be extended without loss of support can be potentially maximal. However still we can obtain non-maximal patterns in the output and some patterns can be generated more than once. Therefore we keep a list of patterns written to output so far. We check any newly generated pattern(even if it can be further extended) with the list and if the list contains more specific pattern with the same occurrences we simply discard the new pattern. The search for new patterns is organized so that any maximal pattern *p* is written to output before any non-maximal patterns less specific than *p*. In this way we never need to remove pattern already written to the output.

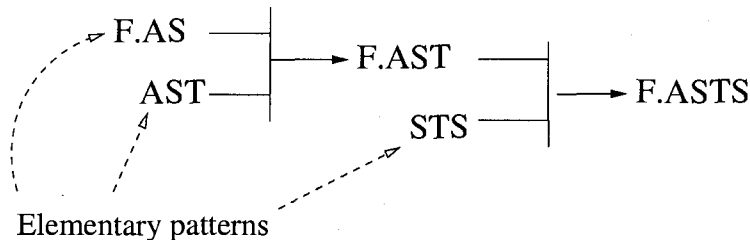


Figure 3.3: Convolution phase of TEIRESIAS

Then order of generating patterns is achieved by careful organization of the depth first search. For this purpose we define prefix and suffix ordering of the set of patterns.

Prefix ordering is defined as follows. Take both patterns and replace all wildcard characters with 0 and other characters with 1. Compare the resulting string lexicographically. The suffix ordering is defined in similar way, except we compare reversed strings. For example *AB.C* is smaller than *AC.B.D* in prefix ordering but it is greater in suffix ordering.

To filtering out non closed patterns, every pattern is assigned a hash value. However, the hash function used by TEIRESIAS can not guarantee that no two closed patterns take the same value. Therefore subsequence matching must be performed to identify non closed patterns. For more details, please refer to [8].

3.1.3 Pattern Growth Method

PrefixSpan [20] method explores prefix-projection in mining of sequential patterns. It greatly reduces the efforts of candidate generation. Moreover, the prefix-projection also reduces the size of projected databases and leads to an efficient processing. Its general idea is to examine

only the prefix subsequence into projected databases. In each projected database, sequential patterns are grown by exploring only local frequent patterns. To further improve the mining, two kinds of database projections are considered.

The algorithm starts by finding all frequent events in the input data. The search space is then divided by partitioning the sequential patterns into subsets having the distinct frequent events as prefixes. This results in the same number of subsets as there are frequent events, the patterns in each subset starting with corresponding event. The subsets can then be mined by constructing corresponding projected databases and mine each recursively as follows. First, the database is scanned in order to find frequent events that can be assembled to the last element or added as such to the end of the prefix to form a sequential pattern. Each such pattern is then output and a respective new prefix-projected database is constructed and explored similarly. The process ends when no new frequent sequences can be generated.

There are two optimizations of the process. The cost of constructing the projected databases can be cut by using bi-level projection, which reduces the number and the size of the projected databases. Secondly, the pseudo-projection can be used to reduce the projection costs when a projected database can be held in the main memory.

PrefixSpan [20] determines all frequent patterns and induces a bottom-up, depth-first search strategy. PrefixSpan is proved much more efficient than GSP [20] but does not support concept graphs, and there is no straightforward way of incorporating them as far as we know.

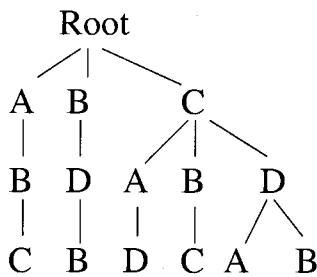
3.2 Efficient Support Counting

The general approach of pattern discovery of algorithms is candidate pattern generation and test. That is they hypothesize possible pattern in turn and verify whether its support exceeds the predefined user threshold. The verification process is critical because it has to be performed for each candidate pattern. The second category algorithms propose elegant data structure for fast support counting to improve the efficiency. PSP [14] used prefix tree, while SPADE [30] and SPAM [3] proposed vertical representation of database for this purpose.

3.2.1 PSP

The PSP algorithm follows the same approach as GSP. However, the authors use prefix tree instead of hash tree, which is used in GSP, as their internal data structure for support counting. Figure 3.4 shows the prefix tree and hash tree used by of PSP and GSP.

PSP Candidate Tree



GSP Candidate Tree

Hash Function: $h(A)=h(B)=0$; $h(C)=h(D)=1$

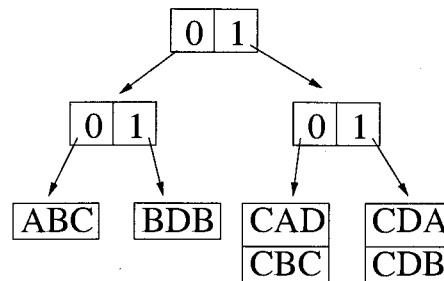


Figure 3.4: Prefix tree vs. Hash tree

During the support counting phase in GSP, for all length k subsequence of each database sequence, the hash tree is navigated until reaching a leaf node storing several candidate patterns. Then each candidate pattern is examined for a match. However, using the prefix tree in PSP, the search for a length k subsequence is terminated when for any $j < k$, no length j subsequence is present in the prefix tree. Once the leaf node is reached, the only operation to perform is to increment its support value. Other benefits of using prefix tree are that it requires less storage space and improves efficiency during the candidate generation phase. This approach proves to be more efficient than GSP in [14]. Since PSP adopts the level wise candidate generation and test approach, it suffers the same problems as GSP.

3.2.2 SPADE

All algorithms explained so far work on a horizontal database format. In horizontal format the database contains a list of sequences (sid), each with its own list of elements (eid). In SPADE, the database layout is transformed from horizontal to vertical. That is, the database is reorganized so that, each element is associated with a id-list. Id-list contains the sequence id (sid) and position id (pid). The (sid, pid) pairs of an element record sequences in which it occurs and where it occurs in that sequence. For example, figure below shows how an

original database can be transformed to a vertical representation. From this database format, it is easy to generate the id-list of a new generated candidate pattern by intersecting of the id-lists of two its generating sequences. For example, using the vertical representation shown in figure below, pattern BA will have id-list $\langle (2,3), (3,4) \rangle$, CB will have id-list $\langle (2,2), (3,3) \rangle$.

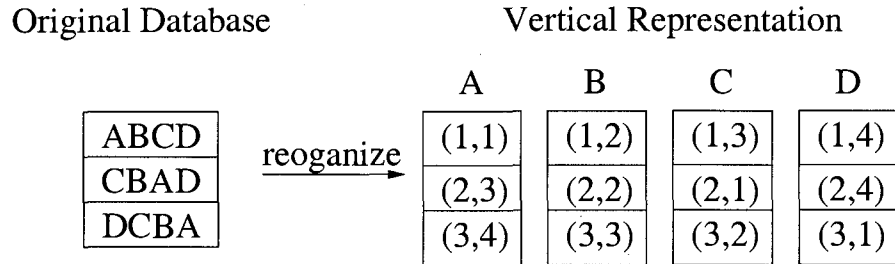


Figure 3.5: Horizontal vs Vertical Representation of Database

Both breadth first and depth first search can be applied to find out frequent patterns. The draw back of depth first search is that we only use two id-lists to generate the next level's patterns and do not fully take advantage of the a priori heuristic, we may unnecessarily generate a larger set of candidate patterns, some of which may not even exist in the database. The draw back of breadth first search is that the huge number of candidate patterns can not be hold in the main memory. However, the author proposed to decompose the search space into equivalence classes such that each equivalence class can be processed independently. For additional details on splitting the search space, we refer the reader to [30].

3.2.3 SPAM

The recently proposed SPAM [3] method is similar to SPADE. The major difference is that bitmaps are used instead of id-lists for the vertical representation. Experiments demonstrate that SPAM outperforms all other methods for finding all frequent sequential patterns. Both, SPADE and SPAM do not support concept graphs, but they can be extended in a straightforward way by extending the alphabet to include the concepts from a concept graph. This will be discussed in more detail later.

For each item in the database, a bitmap is created. And each bitmap has a bit corresponding to each element in the database. If an element i appears in sequence j at position k , then the bit corresponding to sequences j at position k of the bitmap for item i is set

to 1; otherwise the bit is set to 0. The figure below shows how original database can be reorganized to its vertical representation.

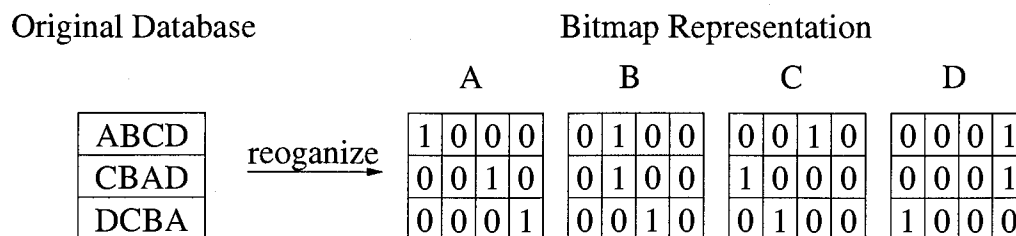


Figure 3.6: Horizontal vs Bitmap Representation of Database

Efficient counting of support is the main advantage of the vertical bitmap representation of data. Support counting for each pattern then becomes a simple check as to whether there exists a 1 in the corresponding sequence. To create the bitmap of a newly generated pattern, use bit operations *shift* and *and* will be used. For example, in order to get the bitmap of pattern *BA*, we first *shift* bitmap of *B* 1 bit to the right, and then *and* with the bitmap of *A*. The figure below shows the procedure.

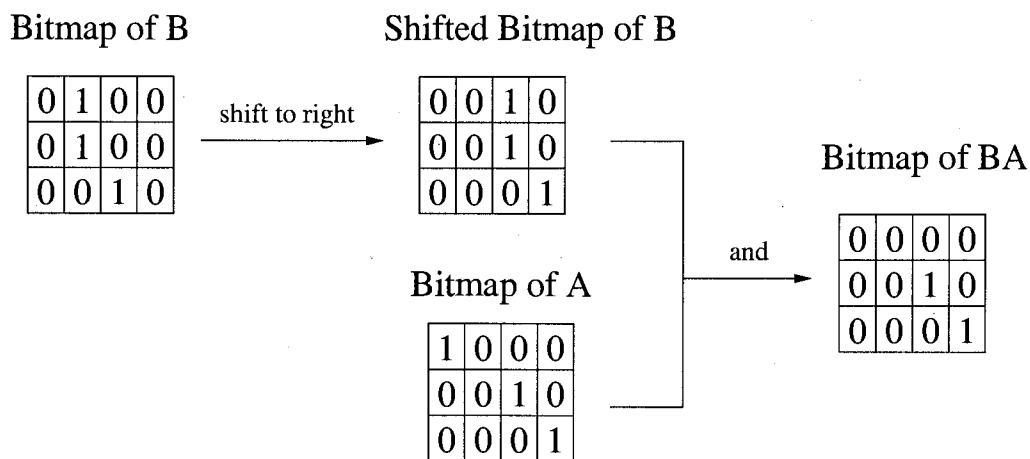


Figure 3.7: Generation of a new bitmap

3.3 Version Space

Besides previous surveyed work done in KDD area, version space, which has been widely studied in Machine Learning community, is closely related to our work. Roughly speaking,

the version space is the subset of hypothesis space consistent with all training examples. Search in version space will either lead from specific to general, driven by positive examples (bottom-up), or from general to specific, driven by negative examples (top-down). The Candidate-Elimination Algorithm combines a bottom-up and a top-down search. In this section, we introduce the concept of version space and the search strategies used to explore it. At the end of this section, we discuss the relationship of the version space to the problem of frequent pattern mining.

Concept learning is one of the central tasks of machine learning. It acquires the definition of a general category given a sample of positive and negative training examples of the category.

The set of items over which the concept is defined is called the set of *instances* X . The concept to be learned is called the *target concept* c . In general, c can be any boolean-valued function defined over the instances X , i.e., $c : X \rightarrow \{0, 1\}$. Training data are instances (records) from an *instance space* X , along with their concept value: $\langle x_1, c(x_1) \rangle, \dots, \langle x_n, c(x_n) \rangle$. Often, $X \subseteq D_1 \times \dots \times D_d$, where D_i is domain of attribute i . Given a set of training data of target concept c , the problem is to hypothesize c . However, due to training data can not cover the whole instance space, inductive learning algorithms can at best guarantee that the output hypothesis fits the target concept over the training data. More formally, learner wants to induce *hypotheses* $h : X \rightarrow \{0, 1\}$ from a set of all possible *hypotheses* H such that $h(x) = c(x)$ for all x in D .

Concept learning can be viewed at the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation. The goal of this search is to find the hypothesis that best fits the training examples. However, even for trivial examples, the space of hypotheses is very large. To explicitly represent the structure of this space, even for rather simple concepts, can involve representing millions of possible hypotheses! For example, if we use 5 dimensions where each dimension can take on one of two values than there are 2 to the 5 or 32 possible instances. The size of this power set is 2 to the 32 or 4,294,967,296. This is a large space to explicitly represent and hold in memory.

Tom Mitchell demonstrated in his research [15] [16] [17] how this structure could be efficiently exploited. Essentially he developed a way in which to implicitly represent and use a structured space of hypotheses. The space was called a version space.

A hypothesis h is *consistent* with a set of training examples D of target concept C iff $h(x) = c(x)$ for each $\langle x, c(x) \rangle$ in D . The *version space* is the subset of hypotheses from

H consistent with all training examples.

One obvious way to represent the version space is simply to list all of its members. For example, we first initialize the version space to contain all hypotheses in H , then eliminate any hypothesis found inconsistent with any training example. The version space of candidate hypotheses thus shrinks as more examples are observed, until ideally just one hypothesis remains that is consistent with all the observed examples. If insufficient data is available to narrow the version space to a single hypothesis, then the algorithm can output the entire set of hypotheses consistent with the observed data. This naive approach needs us to exhaustively enumerate all hypotheses in H .

The Candidate-Elimination algorithm works on the same principle as above naive approach. However, it employs a much more compact representation of the version space. In particular, the version space is represented by its general boundary G , which is the set of its maximally general members and its specific boundary S , which is the set of its maximally specific members. Every member of the version space lies between these boundaries. Figure 3.8 shows the algorithm Candidate-Elimination.

The basic idea of Candidate-Elimination algorithm is to start an inductive learning task with two of the possible hypotheses, the most general hypothesis and the most specific hypothesis. Then, positive examples will always be consistent with the most general hypothesis, but will be inconsistent with the most specific hypothesis. Consequently, this most specific hypothesis will be made more general. A negative example will be consistent with the most specific hypothesis, but inconsistent with the most general hypothesis. Consequently, this most general hypothesis will be made more specific. Thus, at any point in the training sequence the learner will maintain two hypotheses; and, the true hypothesis must lie somewhere in the area of the hypothesis space that connects these two hypotheses. If at some point the most general hypothesis is the same as the most specific hypothesis, then the learner has arrived at a unique definition of the concept. Thus, with a method of this sort the definition of a concept is usually not uniquely determined until enough training instances have been observed that force the learner to a unique determination of the correct hypothesis. Thus, without enumerating the whole search space, Candidate-Elimination finds every hypothesis consistent with the training data.

It is interesting to notice the relationships of the version space to the problem of mining frequent patterns, in particular, to the problem of mining most specific frequent patterns which will be formally introduced in chapter 4. First, the general boundary and specific

Algorithm 3.3 Candidate-Elimination Algorithm**Input:** a set of training data**Output:** general boundary G and specific boundary S of the version space**Method:**

1. Function Candidate-Elimination
2. $G \leftarrow$ maximally general hypotheses in H ;
3. $S \leftarrow$ maximally specific hypotheses in H ;
4. **for each** training example $d = \langle x, c(x) \rangle$
5. **if** d is a positive example
6. remove from G any hypothesis that is inconsistent with d ;
7. **for each** hypothesis s in S that is not consistent with d
8. remove s from S ;
9. add to S all minimal generalizations h of s such that
10. (1) h is consistent with d ;
11. (2) some member of G is more general than h
12. remove from S any hypothesis more general than another hypothesis in S ;
13. **if** d is a negative example
14. remove from S any hypothesis that is inconsistent with d ;
15. **for each** hypothesis g in G that is not consistent with d
16. remove g from G ;
17. add to G all minimal specialization h of g such that
18. (1) h is consistent with d ;
19. (2) some member of S is more specific than h
20. remove from G any hypothesis less general than another hypothesis in G ;

Figure 3.8: Candidate-Elimination Algorithm

boundary provide a compact representation of the version space. Every member of the version space lies between these boundaries. In the problem of mining most specific frequent patterns, the set of all most specific frequent patterns is the specific boundary for the set of all frequent patterns. And the set of the maximal generalizations 20 amino acids is the general boundary of the set of all frequent patterns. All frequent patterns lie between these two boundaries. Second, in order to find the general and specific boundary of version space, Candidate-Elimination combines bottom-up and top-down search. It keeps generalizing the specific boundary and silicifying the general boundary to reach the final boundaries. Our approach ToMMS, in order to find the set of most specific frequent patterns, keeps generalizing the patterns found to be infrequent, i.e. keep generalizing the boundary until the patterns on it are frequent. Although there are some interesting relationships between the version

space and our problem, they are conceptually different problems and the detail approaches to solve them, as a consequence, are different. We will discuss the problem of mining most specific frequent pattern in chapter 4.

Chapter 4

The Algorithm

This chapter provides an in-depth discussion of mining most specific frequent patterns in biological data. A precise definition of the problem is given. A new pattern discovery algorithm called ToMMS is introduced and its design and its efficiency is discussed in detail. Section 4.1 introduces notions of most specific frequent patterns and the problem definition. Section 4.2 presents ToMMS, a top-down method for mining such patterns. Section 4.3 shows an example and analyzes the search space of the concrete patterns. Concept graphs blow up the search space. Their efficient treatment is discussed in section 4.4. How to determine the maximal length of frequent patterns is discussed in section 4.5. The correctness of ToMMS is shown in section 4.6. Section 4.7 discusses the implementation of ToMMS.

4.1 Terminologies and Problem Definition

In this subsection, we introduce terminologies will be used and describe the problem of mining most specific frequent patterns.

Let Σ be a set of symbols, the *alphabet*. The database D consists of n *database sequences* $s = s_1 s_2 \cdots s_l$ with $s_i \in \Sigma$ for $1 \leq i \leq l$, l is the length of s . For protein sequence databases, e.g., $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$, i.e. the one-letter codes of the 20 amino acids.

Let Ω be a set of *concepts*, $\Sigma \cap \Omega = \emptyset$. For example, concepts may represent subsets of the set of all amino acids with the same physico-chemical properties. In this thesis, small letters are used to denote the concepts. For example, s may represent the property small, p the property polar.

Based on Σ and Ω , we define a *concept graph* Γ as a directed acyclic graph $\Gamma \subseteq (\Sigma \cup \Omega) \times \Omega$, where a directed edge $(X, Y) \subseteq (\Sigma \cup \Omega) \times \Omega$ represents an *is-a* relationship “ X is-a Y ”, i.e. the concept Y subsumes the concept or symbol X . Usually concept graphs are provided by biology experts. Figure 4.1 shows a simple concept graph. We will use it as the running example in this paper. The *distance* of two vertexes v_i and v_j in Γ , denoted by $dist(v_i, v_j)$, is the number of the edges of the shortest path between v_i and v_j .

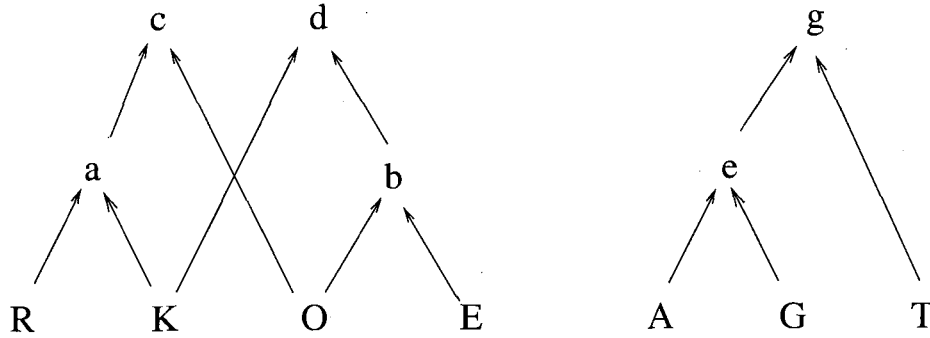


Figure 4.1: Example of a concept graph

The *level* of a symbol $X \in \Sigma \cup \Omega$ w.r.t. concept graph Γ is defined as following:

1. $\forall X \in \Sigma : level(X) = 0$;
2. $\forall X \in \Omega : level(X) = \max\{dist(X, Y) | Y \in \Sigma\}$.

In figure 4.1, for example, $level(A) = 0$, $level(d) = 2$.

A concept graph Γ induces a relation *more specific*, denoted by $<_{\Gamma}$, on as follows:

$$X <_{\Gamma} Y : (X, Y) \in \Gamma.$$

We can extend $<_{\Gamma}$ to \leq_{Γ} by adding $\forall e \in \Sigma \cup \Omega : e \leq_{\Gamma} e$.

The relation \leq_{Γ} has the following properties:

1. $\forall e, f \in \Sigma \cup \Omega : (e \leq_{\Gamma} f) \wedge (f \leq_{\Gamma} e) \Rightarrow (e = f)$ (antisymmetric);
2. $\forall e, f, g \in \Sigma \cup \Omega : (e \leq_{\Gamma} f) \wedge (f \leq_{\Gamma} g) \Rightarrow (e \leq_{\Gamma} g)$ (transitive).

Thus \leq_{Γ} is a partial order. Intuitively, if $X \leq_{\Gamma} Y$, then Y subsumes X , i.e. Y is either identical to X or Y generalizes X . Using concept graph in figure 4.1, for example, $A <_{\Gamma} a$, $a <_{\Gamma} c$, $sp(A, C) = A$.

The *most specific common predecessor* Z of two symbols or concepts $X, Y \in \Sigma \cup \Omega$, denoted by $mscp(X, Y) = Z \in \Sigma \cup \Omega$, is defined by the following two properties:

1. $(X \leq_{\Gamma} Z) \wedge (Y \leq_{\Gamma} Z)$;
2. $\forall Z' \neq Z : (X \leq_{\Gamma} Z') \wedge (Y \leq_{\Gamma} Z') \Rightarrow \neg(Z' <_{\Gamma} Z)$.

A *pattern* is a sequence $p = e_1 e_2 \cdots e_l$ with $e_i \in \Sigma \cup \Omega$, and l is the *length* of the pattern, denoted by $length(p)$. A pattern $p = e_1 e_2 \cdots e_j$ is a *subsequence* of a pattern $p' = e'_1 e'_2 \cdots e'_i$, if $\exists a, 1 \leq a \leq i : \forall b, 1 \leq b \leq j : e'_{a+b-1} = e_b$. For example, CD is a subsequence of CDE . A *concrete pattern* contains only symbols from Σ , for example CDE . A non-concrete pattern such as CbE is called a *generalized pattern*.

We extend the definition of the level and the partial order \leq_{Γ} from single symbols to whole patterns as follows.

The *level* of a pattern w.r.t. graph Γ is the sum of the levels of all elements of the pattern in the graph Γ , i.e. the level of a pattern $p = e_1 e_2 \cdots e_l$ is:

$$level(p) = \sum_{i=1}^l level(e_i).$$

For example, using the concept graph shown in figure 1, pattern CDE has level 0, pattern cbE has level 3.

A pattern p' is called to be *more specific* than a pattern p if p' can be generated from p by specializing one or more of its elements (using the concept graph) and / or by extending it by one or more elements. In this case, we also say that pattern p *generalizes* pattern p' or p is more general than p' . More formally, a pattern $p' = e'_1 e'_2 \cdots e'_i$ is called to be *more specific* than a pattern $p = e_1 e_2 \cdots e_j$ ($i \geq j$), denoted by $p' <_{\Gamma} p$, if

1. $p' \neq p$, and
2. $\exists a, 1 \leq a \leq i : \forall b, 1 \leq b \leq j : e'_{a+b-1} \leq_{\Gamma} e_b$.

For example, $aED <_{\Gamma} abD$ w.r.t. the concept graph shown in figure 1. Note that a pattern is not more specific than itself. We again extend $<_{\Gamma}$ to \leq_{Γ} by adding: for all p over $\Sigma \cup \Omega : p \leq_{\Gamma} p$. From the definition, \leq_{Γ} defined on patterns is also a partial order.

A database sequence s *supports* a pattern p if s is more specific than or equal to p , i.e. $s \leq_{\Gamma} p$. Sequence s is called a *supporting sequence* of p . A subsequence p' of s with properties:

1. $length(p') = length(p)$, and
2. $p' \leq_{\Gamma} p$,

is a *supporting pattern* of p . The *support* of a pattern p in database D , denoted by $sup(p)$, is the number of supporting sequences of p in D . The set of all supporting sequences of a pattern p is denoted by $SupSeqs(p)$. Assuming the concept graph of figure 1, for example, sequence CDE supports all of the following patterns: CDE , CbE , CDb , dD , Cb , etc. The supporting patterns are CDE for the first three of these patterns and CD for the remaining two ones.

A pattern is *frequent* in database D if its support is at least equal to minimum support δ , a user supplied threshold. A frequent pattern p is called a *most specific frequent pattern* if there is no frequent pattern p' which is more specific than p . More precisely, the set of all most specific frequent patterns w.r.t. minimum support δ and concept graph Γ is the set S of patterns with the following properties: $\forall p \in S$

1. $sup(p) \geq \delta$, and
2. $\forall p' \neq p : sup(p') \geq \delta \Rightarrow \neg(p' <_{\Gamma} p)$.

The following theorem shows that the set of all frequent patterns can be derived from the set of all most specific frequent patterns by generalizing one of these patterns.

Theorem 4.1.1 *Let S be the set of all most specific frequent patterns in database D w.r.t. minimum support δ and concept graph Γ . Then, $\forall p : sup(p) \geq \delta \Rightarrow \exists q \in S : q \leq_{\Gamma} p$.*

Proof (1) If $p \in S$, we are done. (2) If $p \notin S$, then $\exists p_1 : (sup(p_1) \geq \delta) \wedge (p_1 \leq_{\Gamma} p)$ because p is not most specific. The same reasoning can now recursively be applied to p_1 , i. e. either case (1) or case (2) applies to p_1 . We obtain a (potentially infinite) chain of patterns: $\dots p_n \leq_{\Gamma} \dots \leq_{\Gamma} p_2 \leq_{\Gamma} p_1 \leq_{\Gamma} p$ with $sup(p_i) \geq \delta$, $1 \leq i \leq n$. This chain must be finite, since there is only a finite number of frequent patterns in D w.r.t. δ (because a frequent pattern generalizes all its supporting database sequences and, therefore, cannot be longer than these sequences). i.e., the case (1) must eventually apply to $q = p$ and we obtain a pattern $q \in S$ with $q = p_n \leq_{\Gamma} \dots \leq_{\Gamma} p_2 \leq_{\Gamma} p_1 \leq_{\Gamma} p$. Because of the transitivity of \leq_{Γ} , we conclude $q \leq_{\Gamma} p$. ■

Using the above definitions we succinctly describe the problem addressed by ToMMS as follows:

Problem Definition Given database $D = \{s_1, s_2, \dots, s_n\}$ of sequences over alphabet Σ , concept graph Γ and minimum support δ , construct the set of all most specific frequent patterns with support at least δ in D w.r.t Γ .

4.2 ToMMS

Given a sequence database D , a concept graph Γ and a minimum support value δ , the set of all most specific frequent patterns shall be discovered. In this subsection, we present the algorithm ToMMS which efficiently mines all such patterns.

ToMMS is based on the following key observations:

- Frequent concrete patterns must be subsequences of data sequences.
- More specific patterns should be generated before less specific ones in order to avoid the unnecessary generation of frequent patterns which are not most specific.

To exploit these observations, instead of using the traditional bottom-up approach, we propose a top-down approach to explore the search space. We start with all subsequences of the data sequences (of some specified length maximum-length) and continue to shorten them by one element until they become frequent. Furthermore, infrequent patterns are also generalized using the concept graph until they become frequent.

Figure 4.2 shows the algorithm ToMMS using the notations introduced in Table 4.1.

Set	Contains
$MSFP_i$	Most Specific Frequent Patterns of length i
CP_i	Concrete Patterns of length i
ICP_i	Infrequent Concrete Patterns of length i
FCP_i	Frequent Concrete Patterns of length i
GP_i	Generalized Patterns of length i
IGP_{ij}	Infrequent Generalized Patterns of length i at level j
FGP_{ij}	Frequent Generalized Patterns of length i at level j

Table 4.1: Notations

In the function *Generalize*, the infrequent concrete patterns of length i are minimally generalized until they become frequent. In order to perform a minimal generalization,

Algorithm 4.2.1 Algorithm ToMMS**Input:** a sequence database D , concept graph Γ and minimum support value δ **Output:** the set of most specific frequent patterns in D **Method:**

1. Function ToMMS(D, Γ, δ)
2. /*Returns maximal length of most specific frequent pattern*/
 maxlength \leftarrow Maxlength (D, Γ, δ);
3. $i \leftarrow$ maxlength;
4. /*Return all subsequences of length maxlength in D^* */
 $CP_i \leftarrow$ Preprocess($D, \text{maxlength}$);
5. **while** $CP_i \neq \emptyset$ **do**
6. $MSFP_i \leftarrow \{c | c \in CP_i, c \text{ is frequent}\}$;
7. $FCP_i \leftarrow FCP_i \cup \{c | c \in CP_i, c \text{ is frequent}\}$;
8. $ICP_i \leftarrow \{c | c \in CP_i, c \text{ is infrequent}\}$;
9. $MSFP_i \leftarrow MSFP_i \cup \text{Generalize}(ICP_i, FCP_i)$;
10. $MSFP_i \leftarrow \text{Subseqcheck}(MSFP_i)$;
11. $CP_{i-1} \leftarrow \text{Shorten}(ICP_i)$;
12. $FCP_{i-1} \leftarrow \text{Shorten}(FCP_i)$;
13. $i \leftarrow i-1$;
14. **end-while**
15. **return** $\bigcup_i MSFP_i$;

Figure 4.2: The ToMMS Algorithm

pairs of two infrequent patterns are matched and generalized using the concept graph. For example, using the concept graph of figure 1, the infrequent candidates RQK and REK can be minimally generalized to RbK . The function *Generalize* returns all most specific frequent generalized patterns of length i . The details of generalizing infrequent patterns will be discussed in section 4.4.

The function *Shorten* shortens all patterns contained in the input set by 1 element. For example, the set $\{RKQ, KQA\}$ is shortened to the set $\{RK, KQ, QA\}$.

For fast support counting of pattern p , we use bitmaps to record $SupSeq(p)$. Support counting for the patterns in CP_i is not explicitly mentioned in the pseudo-code, because it is performed implicitly when inserting new patterns into the data structure CP_i . Whenever a newly generated pattern already exists in CP_i , the corresponding support count is updated according to the supporting sequences of the newly generated pattern. This method is correct since all the information of the original database is contained in the sets of FCP_i and ICP_i . Thus, we do not need to scan the original database for support counting. A

similar method can be applied to count the support of generalized patterns (see section 4.4).

For the purpose of avoiding redundancy, the function *Subseqcheck* is used to remove those frequent patterns which are not most specific.

The function *Maxlength* returns the maximum length of frequent patterns, which is also the maximum length of most specific frequent patterns. We will discuss it further in section 4.5.

4.3 Concrete Patterns: Example and Analysis

In this subsection, we show an example of the search space of concrete patterns and analyze its complexity. The problem of mining most specific general patterns is the crucial part of ToMMS and will be elaborated in section 4.4.

Let the data sequences be the sequences shown in figure 4.3, which will be used as our running example throughout this thesis.

RKQA
KRQP
KKE

Figure 4.3: Dataset of running example

Now let us assume that max-length is 4. ToMMS starts from the subsequence of length 4, e.g. the data sequence *RKQA*, and checks the support of this sequence. If it turns out to be infrequent, all subsequences of length 3 are generated, i.e. the two subsequences *RKQ* and *KQA*. If any one of them is frequent, we output it and stop generating its subsequences. For example, if *RKQ* turns out to be frequent, we do not check the subsequences *RK* and *KQ*. Figure 4.4 depicts the search space of all concrete patterns generated by sequence *RKQA*.

Suppose the maximum length of the database sequences (not the patterns) is l and the number of the sequences in the database is n , then the maximal number of concrete patterns is $n \cdot l \cdot (l - 1) / 2 = O(n \cdot l^2)$. For typical biological databases, this number is much smaller than the worst case number of candidate patterns for the bottom-up approach which is

$O(|\Sigma|^m)$ where $|\Sigma|$ denotes the size of the alphabet and m is the maximum length of frequent patterns. For example if there are 10 concepts and the length of the longest pattern is 20, then the number of potential frequent patterns is 30^{20} for bottom-up enumeration.

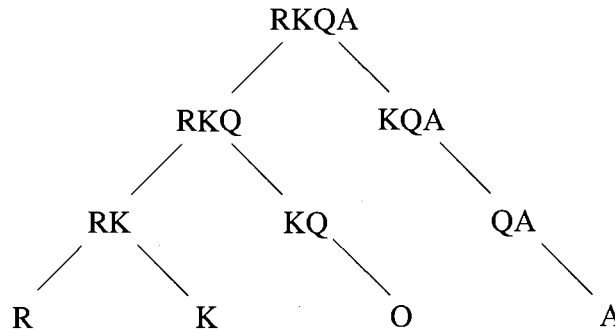


Figure 4.4: Example of search space of concrete pattern

4.4 Incorporating Concept Graphs

Although we have reduced the search effort for concrete patterns, concept graphs blow up the size of the search space dramatically. For example, a single concrete pattern of length l can be generalized in 2^l ways even if we consider the simplest case of a one-level concept tree (instead of a general non-hierarchical concept graph). In this section, we elaborate our method for searching the space of generalized patterns. The correctness of the ToMMS algorithm is shown in section 4.6.

We want to avoid any unnecessary generalizations of infrequent patterns, i.e. generalizations which either do not increase the support of the pattern or create frequent patterns which are not most specific. We achieve this goal by two features:

- We again perform a top-down search, i.e. only infrequent patterns are generalized using the concept graph.
- For such patterns, we apply only a small subset of all possible generalizations, i.e. only generalizations which actually increase the support.

Note that a generalization operation need not necessarily increase the support of a pattern. Our key idea is to generate only patterns which generalize two (not one) infrequent

patterns. Assuming that the sets of supporting sequences of the two input patterns are not proper subsets of each other, the generalized pattern must have a larger support than any of the input patterns. Since the procedure Shorten already considers subsequences of the current infrequent patterns, here we consider only generalizations of two same length patterns.

We define a match of n patterns as a set of all patterns of the same length generalizing all input patterns such that there is no more specific pattern with the same property. More formally, a match \oplus of n patterns p_1, p_2, \dots, p_n , denoted by $S = \oplus(p_1, p_2, \dots, p_n)$, is defined as the set of all patterns p with the following properties:

- (1) $\forall i, 1 \leq i \leq n : \text{length}(p) = \text{length}(p_i)$,
- (2) $\forall i, 1 \leq i \leq n : p_i \leq_{\Gamma} p$,
- (3) $\forall p' \neq p : (p_1 \leq_{\Gamma} p') \wedge (p_2 \leq_{\Gamma} p') \wedge (p_n \leq_{\Gamma} p') \Rightarrow \neg(p' \leq_{\Gamma} p)$.

Such a match does not exist for all pairs of patterns, i.e. the match may be the empty set. The match of two patterns is at most a single pattern as long as the concept graph is tree-structured, i.e. if there is at most one predecessor for any concept or symbol from the alphabet. For non tree-structured concept graphs, however, there may be several matching patterns of two input patterns. For example, using the concept graph of figure 1, $ACE \oplus ADE = \{AcE, AdE\}$, while $ACD \oplus ECD = \emptyset$ since there is no common predecessor for A and E in the concept graph.

The following theorem states that all generalizations of a set of input patterns can be derived from the match of these patterns.

Theorem 4.4.1 *Let $S = \oplus(p_1, p_2, \dots, p_n)$ for pattern p_1, p_2, \dots, p_n over $\Sigma \cup \Omega$. For every pattern p' with the following two properties (1) $\text{length}(p') = \text{length}(p_i), i = 1, 2, \dots, n$ and (2) $(p_1 \leq_{\Gamma} p') \wedge (p_2 \leq_{\Gamma} p') \wedge (p_n \leq_{\Gamma} p')$, there is a pattern $p \in S$ such that $p \leq_{\Gamma} p'$.*

Proof Similar to the proof of Theorem 3.1.1. ■

The next theorem shows that any match of more than two patterns is equivalent to a sequence of pair-wise matches of the same patterns.

Theorem 4.4.2 *Let $p \in \oplus(p_1, p_2, \dots, p_n)$ for patterns p_1, p_2, \dots, p_n over $\Sigma \cup \Omega$, then $\exists p_m \in \oplus(p_1, p_2, \dots, p_{n-1})$ such that $p \in \oplus(p_m, p_n)$.*

Proof If the theorem does not hold, we have $\forall p_m \in \bigoplus(p_1, p_2, \dots, p_{n-1}), p \notin \bigoplus(p_m, p_n)$. That means $\exists p' \neq p, p_m \leq_{\Gamma} p', p_n \leq_{\Gamma} p', p' <_{\Gamma} p$. So we have $\forall i, 1 \leq i \leq n-1 : p_i \leq_{\Gamma} p_m \leq_{\Gamma} p', p_n \leq_{\Gamma} p', p' <_{\Gamma} p$. This is a contradiction to the assumption $p \in \bigoplus(p_1, p_2, \dots, p_n)$ which implies that p is a *minimal* generalization of p_1, p_2, \dots, p_n . ■

The match of two patterns can be efficiently implemented: for each pair of corresponding elements of the two input patterns, determine all most specific common predecessors in the concept graph and form all possible combinations of them. This is obviously correct from the definition of most specific common predecessor and the definition of most specific frequent pattern.

Although the proposed approach of pairwise generalizing infrequent sequences drastically reduces the search space, many redundant matches (patterns) will be generated by a naive adoption of this approach. This is due to the fact that two different pairs of infrequent patterns may yield two matching patterns m_1 and m_2 with $m_1 \leq_{\Gamma} m_2$. While the case of $m_1 = m_2$ is easy to detect, the case of $m_1 <_{\Gamma} m_2$ is much more difficult to determine. To speed-up the necessary tests for $<_{\Gamma}$ relationships among the set of all matches, we exploit the following dependencies between the levels of two patterns and their order w.r.t. $<_{\Gamma}$. First, if two same length patterns p_1 and p_2 have the same level, then neither p_1 is more specific than p_2 , nor p_2 is more specific than p_1 . Second, if a pattern p_1 has a smaller level than another pattern p_2 , then p_1 cannot be more general than p_2 . These two properties are formalized in the following two lemmas.

Lemma 4.4.3 *Let p_1 and p_2 be patterns over $\Sigma \cup \Omega$ with $\text{length}(p_1) = \text{length}(p_2)$. Then the following holds: $\text{level}(p_1) = \text{level}(p_2) \Rightarrow \neg(p_1 <_{\Gamma} p_2) \wedge \neg(p_2 <_{\Gamma} p_1)$.*

Proof Suppose $p_1 = e_1 e_2 \dots e_n$ and $p_2 = e'_1 e'_2 \dots e'_n$. If $\text{level}(e_1) = \text{level}(e'_1), \text{level}(e_2) = \text{level}(e'_2), \dots, \text{level}(e_n) = \text{level}(e'_n)$, we have $\neg(p_1 <_{\Gamma} p_2)$. If for some i , $\text{level}(e_i) < \text{level}(e'_i)$, since $\text{length}(p_1) = \text{length}(p_2)$ and $\text{level}(p_1) = \text{level}(p_2)$, there must be some j , such that $\text{level}(e_j) > \text{level}(e'_j)$. We have $\neg(p_1 <_{\Gamma} p_2)$. The rest part can be proved similarly. ■

Lemma 4.4.4 *Let p_1 and p_2 be patterns over $\Sigma \cup \Omega$ with $\text{length}(p_1) = \text{length}(p_2)$. Then the following holds: $\text{level}(p_1) < \text{level}(p_2) \Rightarrow \neg(p_1 >_{\Gamma} p_2)$.*

Proof Similar to the proof of lemma 4.4.3. ■

Exploiting these two lemmas, we partition the set of all pairwise matches of infrequent patterns of a given length into subsets of patterns of the same level. We consider these subsets separately in ascending order of their level. According to lemma 4.4.3, we do not have to test for $<_{\Gamma}$ relationships among patterns of the same level (the $=$ relationship is still possible). According to lemma 4.4.4, this method guarantees that a generalized pattern is never considered before one of its specializations. Therefore, we can immediately test whether a current candidate pattern generalizes any already discovered frequent pattern of the same length and smaller level (if yes, it is not reported). This means that we do not have to delay the test for $<_{\Gamma}$ relationships among the generalized frequent patterns until we have determined all of them.

Algorithm 4.4.1 Algorithm Generalize

Input: set of infrequent and frequent concrete patterns of length i ICP_i and FCP_i

Output: the set of most specific frequent general patterns of length i

Method:

```

1. Function Generalize( $ICP_i, FCP_i$ )
2.   for  $j=1$  to maxlevel do
3.      $GP_{ij} \leftarrow \{p \mid p=p_m \oplus p_n, p_m, p_n \in ICP_i, j=\text{level}(p)\}$ ;
4.   end-for;
5.   for  $j=1$  to maxlevel do
6.     for each  $p \in FCP_i$  do;
7.       mark  $p'$  where  $p' \in GP_{ij}, p' \geq_{\Gamma} p, j=\text{level}(p)$ ;
8.     end-for;
9.      $IGP_{ij} \leftarrow \{p \mid p \in GP_{ij}, p \text{ is infrequent and unmarked}\}$ ;
10.    for each pair  $(p_a, p_b), p_a \in ICP_i, p_b \in IGP_{ij}$  do;
11.      if  $(\text{SupSeq}(p_a) \not\subseteq \text{SupSeq}(p_b))$ 
12.         $GP_{il} \leftarrow GP_{il} \cup \{p \mid p=p_a \oplus p_b, l=\text{level}(p)\}$ ;
13.      end-for;
14.     $FGP_{ij} \leftarrow \{p \mid p \in GP_{ij}, p \text{ is frequent}\}$ ;
15.    for each  $p \in FGP_{ij}$  do
16.      mark  $p'$  where  $p' \in GP_{ij}, p' \geq_{\Gamma} p, l=\text{level}(p)$ ;
17.    end-for;
18.  end-for;
19.  return  $\bigcup_j FGP_{ij}$ ;

```

Figure 4.5: Function Generalize

Figure 4.5 presents the pseudo-code of the generalization method which takes the sets of all infrequent / frequent concrete patterns of length i and returns the set of all most specific, non-concrete frequent patterns of length i .

The purpose of the marking in function *Generalize* is to avoid the overhead of redundant matching operations. Every time a pattern p turns out to be frequent, all its generalizations must be frequent and cannot be most specific patterns. Therefore, we mark all generalizations of p in higher level generalized pattern sets which will not be generalized further.

We use an example to illustrate the *ToMMS* algorithm. Let us use again the concept graph in table 4.1 and dataset in figure 4.3. We simply start from the length of the longest data sequences (how to find out the maximal length will be discussed in next subsection), i.e. from length 4 and minimum support is 3. We have ICP_4 as shown in table 4.2(a). The ids of the supporting sequences are also given, for example, the supporting sequence of $RKQA$ is sequence 1. Using the concept graph, we pairwise generalize patterns in ICP_4 , i.e. match $RKQA$ and $KRQP$. After matching infrequent concrete patterns, the resulting sets GP_{4i} are shown in table 4.2(b). Please note pattern $aaQg$ has level 4 according to the example concepts. Since both sets of supporting sequences of the patterns in ICP_4 are proper subset of the set of supporting sequences of $aaQg$, we finish length 4 and go to length 3.

- (a) $ICP_4 = \{RKQA:[1], KRQP[2]\};$
- (b) $GP_{41} = \emptyset,$
 $GP_{42} = \emptyset,$
 $GP_{43} = \emptyset,$
 $GP_{44} = \{aaQg:[1,2]\};$

Table 4.2: Example of function *Generalize*: length 4

The patterns in ICP_4 are shortened. Thus we get ICP_3 which is shown in table 4.3 (a). Note the data sequence KKE is also added in ICP_3 . Again, using the concept graph, we pairwise generalize patterns in ICP_3 . The resulted patterns are inserted to corresponding sets according to their levels as shown in table 4.3 (b). Starting from the lowest level nonempty generalized pattern set GP_{32} , we match the infrequent patterns in GP_{32} with the patterns in ICP_3 . For example, RKQ match Kab will get aab . Note that RKQ does not match with aaQ because the set of supporting sequences of RKQ is subset of that of aaQ . All the possible 3 matches (where the sets of supporting sequences are not subsets of each other) of one pattern in ICP_3 and one pattern in GP_{32} yield the same result aab . The result of this generalization is shown in table 4.3(c). Since all patterns in GP_{32} is infrequent, we

go to higher level GP_{33} . Frequent pattern aab is then outputted. Since matching KKE and aQg we get empty set, all generalized patterns are checked and length 3 is done.

- (a) $ICP_3 = \{RKQ:[1], KQA:[1], KRQ[2], RQP[2], KKE[3]\};$
- (b) $GP_{31} = \emptyset,$
 $GP_{32} = \{aaQ:[1,2], aKb:[1,3], Kab:[2,3]\},$
 $GP_{33} = \{aQg:[1,2]\};$
- (c) $GP_{31} = \emptyset,$
 $GP_{32} = \{aaQ:[1,2], aKb:[1,3], Kab:[2,3]\},$
 $GP_{33} = \{aQg:[1,2], aab:[1,2,3]\}$

Table 4.3: Example of function Generalize: length 3

We then proceed length 2 patterns. Patterns in ICP_3 are shortened to length 2 and inserted to ICP_2 as shown in table 4.4 (a). Perform pairwise matching operation on all infrequent length 2 concrete patterns, we get general patterns and their current set of supporting sequences shown in 4.4 (b). Since no pattern in GP_1 is frequent, we generalize them against all infrequent concrete patterns. After this, some patterns' supporting sequences are updated, e.g. patterns Ka and aa . We output Ka as frequent patterns, and mark all its generalizations: aa , Kc , and ac , as shown in 4.4 (c). We continue generalize higher levels and mark those generalizations of frequent patterns. The frequent patterns reported by function Generalize are shown in 4.4 (d). Since pattern ab is a subsequence of previously reported most specific frequent pattern aab , it is discarded.

- (a) $ICP_2 = \{RK:[1], KQ:[1], QA:[1], KR[2], RQ[2], QP[2], KK[3], KE[3]\};$
- (b) $GP_{31} = \{aK:[1,3], aQ:[1,2], Kb:[1,3], Ka:[2,3]\},$
 $GP_{32} = \{aa:[1,2], Rc:[1,2], Rd:[1,2], Kc:[1,2], Kd:[1,3], Qg:[1,2], ab:[2,3]\},$
 $GP_{33} = \{ad:[1,2,3], ac:[2,3]\};$
- (c) $GP_{31} = \{aK:[1,3], aQ:[1,2], Kb:[1,3], Ka:[1,2,3]\},$
 $GP_{32} = \{aa:[1,2,3] \times, Rc:[1,2], Rd:[1,2], Kc:[1,2] \times, Kd:[1,3], Qg:[1,2], ab:[1,2,3]\},$
 $GP_{33} = \{ad:[1,2,3], ac:[1,2,3] \times\};$
- (d) $MSFP_2 = \{Ka:[1,2,3], ab:[1,2,3], ad:[1,2,3]\};$ (before subsequence checking)

Table 4.4: Example of function Generalize: length 2

Note that when we try to filter out non most specific frequent pattern p , we do not need to check whether a specification of p with the same length, p' , is a subsequence of

previously reported patterns. For example, for pattern Ka , we do not need to check whether its specification KK is a subsequence of already reported patterns. Its correctness will be discussed in section 4.6. In this way, we can efficiently identify redundant patterns. Since to check the generalization relationship is much harder than subsequence checking.

4.5 Determining the Maximum Length

In order not to miss some most specific frequent patterns, ToMMS needs to first determine the maximum length of (most specific) frequent patterns, and then starts its top-down search from all data subsequences of this maximum length. The maximum length can be calculated in a bottom-up manner by performing a binary search using the ToMMS algorithm as follows:

- We start with a current length of 1, the minimum maximum length. In the first phase, we keep doubling the current length and apply a simplified ToMMS algorithm with $maxlength = currentlength$ until we find a current length without any frequent patterns.
- In a second phase, we perform a binary search on this interval of possible maximum length values applying again the simplified ToMMS algorithm until we find a length l such there is a frequent pattern of length l but no frequent pattern of length $l + 1$.

The simplified ToMMS algorithm does not determine the actual maximum length and does not consider shorter patterns. Instead, it searches only for frequent patterns of exactly the length specified by its parameter value. Furthermore, the modified *ToMMS* algorithm does not have to find all most specific frequent patterns but stops as soon as the first such pattern has been discovered. The simplified ToMMS algorithm will possibly be called several times (for too large length values) without finding any frequent pattern of this given length, but these runs are relatively inexpensive because of the following reason. If no pattern is frequent of length l , it means that very few generalized patterns will be generated by matching infrequent concrete patterns. And the major overhead of ToMMS is matching infrequent concrete patterns with infrequent generalized patterns. To conclude, the simplified ToMMS and the proposed method for determining the maximum length are quite efficient which will also be confirmed by our experimental evaluation.

4.6 Correctness of ToMMS

Based on the above lemmas and theorems, we will now show the correctness of ToMMS, i.e. we will show that ToMMS reports all most specific frequent patterns theorem 4.6.1 and only most specific frequent patterns are reported theorem 4.6.2.

Theorem 4.6.1 *ToMMS reports all most specific frequent patterns in database D w.r.t. concept graph Γ and minimum support δ .*

Proof 1) Let p be a concrete most specific frequent patterns in database D w.r.t. concept graph Γ and minimum support δ . Let S denote the set of all (consecutive) subsequences of all database sequences in D . ToMMS generates the elements of S in descending order of length and stops shortening a pattern (subsequence) if it is frequent. Since p is a concrete most specific frequent pattern, and no supersequence of p is frequent. Thus, p will be generated and reported by ToMMS.

2) Let p be a non-concrete most specific frequent patterns in database D w.r.t. concept graph Γ and minimum support δ . Without loss of generality, let $SupSeq(p) = \{s_1, s_2, \dots, s_m\}, m \geq \delta$. From each supporting sequence of p , we choose a supporting pattern of p and construct a set of supporting patterns $\{p_1, p_2, \dots, p_m\}$ of p . (i) From theorem 4.4.1 we know that there is a $q \in \bigoplus(p_1, p_2, \dots, p_m)$ with $q \leq_{\Gamma} p$. q is frequent since it has support $m \geq \delta$. Since p is most specific, we conclude that $q = p$, i.e. $p \in \bigoplus(p_1, p_2, \dots, p_m)$. According to theorem 4.4.2, any match of more than two patterns is equivalent to a sequence of pair-wise matches of the same patterns. So p will be generated by pair-wisely matching p_1, p_2, \dots, p_m . (ii) Furthermore, $\forall i, 1 \leq i \leq m, p_i$ is infrequent because of the following reason. Since all p_i are concrete patterns, $\forall 1 \leq i \leq m : p_i \leq_{\Gamma} p$, i.e. $\forall 1 \leq i \leq m : sup(p_i) < sup(p)$. p is most specific, therefore all p_i are infrequent. Because of (i) and (ii), $\bigoplus(p_1, p_2, \dots, p_m)$ will be constructed and this generates p . ■

Theorem 4.6.2 *All patterns reported by ToMMS are most specific frequent patterns in database D w.r.t. concept graph Γ and minimum support δ .*

Proof (1) p is frequent. ToMMS reports only patterns which have been tested and found to be frequent.

(2) p is most specific. First we show that ToMMS never generates a pattern $q, q >_{\Gamma} p$ before pattern p . If $length(q) < length(p)$, since ToMMS generates patterns in descending

order of length, q will not be generated before p . If $length(q) = length(p)$, since ToMMS generates same length patterns in ascending order of level, q will not be generated before p . According to lemma 4.4.3 and lemma 4.4.4, this implies that more specific patterns are always generated before their generalizations. Second, a pattern is only reported if no more specific pattern (of larger length) has been previously reported. Therefore, any pattern reported by ToMMS is most specific. ■

In the second last sentence of above proof, we said that a pattern is only reported if no more specific pattern of larger length has been reported. However, to verify whether there is a more specific pattern of larger length will become harder when the number of reported patterns grows. For example, for pattern p , all its specifications have to be checked against all reported patterns. The following lemma gives us a more efficient way to remove redundant patterns. It says performing subsequence checking is enough to filter out redundant patterns.

Lemma 4.6.3 *For any pattern p reported by function *Generalize*, if p is not a subsequence of any previously reported patterns, p is a most specific frequent pattern.*

Proof Since p is reported by function *Generalize*, from the proof of theorem 4.6.2, we know that all its specifications with same length are infrequent. Thus any longer pattern, q , containing one of p 's specification as its subsequence is infrequent. Such pattern would not have been reported previously. ■

4.7 Efficient Implementation of the Match Operation

The major data structure used to store the sets of patterns is a modified prefix tree, one separate prefix tree for each of the sets listed in table 1. Each path of a prefix tree represents a pattern. For each leaf of the tree, we attach the bitmap of the set of supporting sequences of the pattern represented by the corresponding path. The support of a pattern is updated every time when a new pattern is inserted to the tree. Figure 4.6 shows an example of the modified prefix tree which stores the set of infrequent concrete patterns of length 3, i.e. ICP_3 , from previous example.

The major overhead of ToMMS is using infrequent concrete patterns to match (generalize) infrequent generalized (or concrete) patterns of different levels. In order to optimize the

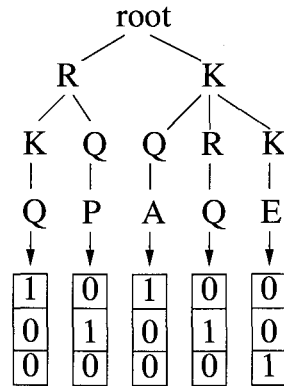


Figure 4.6: Example of modified prefix tree

match operation, we perform prefix tree against prefix tree generalization instead of one pattern against one pattern generalization. Infrequent concrete patterns and generalized patterns of different levels are indexed by separate prefix trees. When using infrequent patterns to match generalized patterns with same level, we perform breadth first search of the two corresponding prefix trees. At each pair of the interior nodes of the two trees, if the two symbols represented by the interior nodes have common predecessors, we continue checking children of the two nodes. Otherwise, the whole subtrees of the two nodes pruned, because the two patterns, presented by the branches of trees, can not be matched to get a more general pattern. The figure 4.7 shows the pseudo code of the procedure of tree against tree generalization.

However, by using the tree against tree generalization, we will generate more patterns than stated in the pseudo code of `Generalize`, because we can not check whether the set of supporting sequences of p is proper subset of the set of supporting sequence p' . But this will not change the correctness of `ToMMS`. There are simply more pair of patterns will be matched. But as a trade off, using tree against tree generalization improve the performance a lot.

Algorithm 4.7.1 Algorithm Gen_tree

Input: a prefix tree storing the set of infrequent concrete patterns and a prefix tree storing set of infrequent general patterns

Output: the set of patterns which are generated by pairwise generalize infrequent concrete patterns and general patterns

Method:

```

1. Function Gen_tree ( prefixtree root1, prefixtree root2)
2.   entry.ptr1 ← root1;
3.   entry.ptr2 ← root2;
4.   entry.pattern ← empty string;
5.   push entry to stack s;
6.   while s is not empty do;
7.     entry ← s.pop;
8.     entry.ptr1 ← entry.ptr1.firstchild;
9.     entry.ptr2 ← entry.ptr2.firstchild;
10.    if (entry.ptr1 = NULL) do
11.      update SupSeq(entry.pattern) and
12.      insert entry.pattern into GPi, where i=level(entry.pattern);
13.    end-if;
14.    while entry.ptr1 ≠ NULL do
15.      while entry.ptr2 ≠ NULL do
16.        for each most specific common predecessors, en,
17.        of entry.ptr1.element and entry.ptr2.element do
18.          entry.pattern ← entry.pattern+en;
19.          /*concatenate en to the entry.pattern*/
20.          push entry to stack s;
21.        end-for;
22.        entry.ptr2 ← entry.ptr2.sibling;
23.      end-while;
24.      entry.ptr1 ← entry.ptr1.sibling;
25.    end-while;
26.  end-while;

```

Figure 4.7: Function Generalize

Chapter 5

Experimental Evaluation

5.1 Datasets

We performed experimental evaluations on three different protein sequence datasets. Two of them, the outer membrane protein dataset and the non outer membrane protein dataset, were produced by the Department of Molecular Biology and Biochemistry at Simon Fraser University, as part of our collaborative efforts to tackle the outer membrane protein identification problem [22]. For proper functioning, a protein has to be transported to the correct intra- or extra-cellular compartments in a soluble form or attached to a membrane; hence the cellular location of a protein sequence plays a key role with regard to its functions. In particular, proteins attached to the outer membrane of cells fulfill a number of tasks that are very crucial, such as solute and protein translocation as well as signal transduction. Outer membrane proteins - which are exposed on the surface of the cell - represent potential drug and vaccine targets, and the ability to identify such potential targets from sequence information alone would allow researchers to quickly prioritize a list of proteins for further study. The dataset was created by extracting all Gram-negative proteins with an annotated subcellular localization site from the SWISSPROT database [5]. The annotated localization sites were then confirmed through a manual search of the literature, and those proteins with an experimentally verified localization site were added to the dataset. The third dataset is the yeast (*Saccharomyces cerevisiae*) protein data set available at http://www.maths.uq.edu.au/fc/datasets/yeast_SC_gb117/yeast_dataset.gb117.html. It is constructed from gene data contained in GenBank [4] release 117. The most important properties of these three datasets are shown in the table 5.1.

Data	Sequence Num.	Min. Length	Max. Length	Ave. Length
OM	417	91	3705	579
non-OM	620	68	1553	392
Yeast	393	15	1859	256

Table 5.1: Description of the evaluation datasets.

We use a typical concept graph of amino-acids as used in [19]. The table below shows the sets of amino acids corresponding to the concepts of this concept graph. These concepts represent the relevant physio-chemical properties of amino-acids.

Physio-chemical properties	Amino Acids
Small	AG
Small hydroxyl	ST
Basic	KR
Aromatic	FWY
Basic	HKR
Small hydrophobic	ILV
Medium hydrophobic	ILMV
Acidic/amid	EDNQ
Small polar	AGPST

Table 5.2: Concepts of amino acids and corresponding physio-chemical properties.

5.2 Design of the Experiments

We chose the competitors of ToMMS as follows. SPAM [20] has been shown to outperform all other algorithms for sequential pattern mining in the data mining community. We implemented SPAM in a modified version to incorporate concept graphs. Basically, the alphabet is extended by the concepts. Thus the pattern language of SPAM is modified to be the same as that of ToMMS. We did not modify SPAM to filter out non most specific patterns, since this would require expensive post-processing due to the fact that SPAM does not necessarily generate most-specific patterns before more general ones. The other algorithm we compared is Teiresias [21] which can be downloaded from <http://cbcsrv.watson.ibm.com/download.phtml.html>.

To the best of our knowledge, this is the best sequence mining algorithm in the bioinformatics community.

All the experiments were performed on a 1.7 GHz Pentium PC machine with 1 Gigabytes main memory, running Microsoft Windows 2000. All the programs are written in Microsoft/Visual C++6.0.

5.3 Experimental Results

Figures 5.1 to 5.3 show the runtimes of ToMMS, Teiresias and SPAM on the three test datasets at different minimum support values using the concept graph from [9].

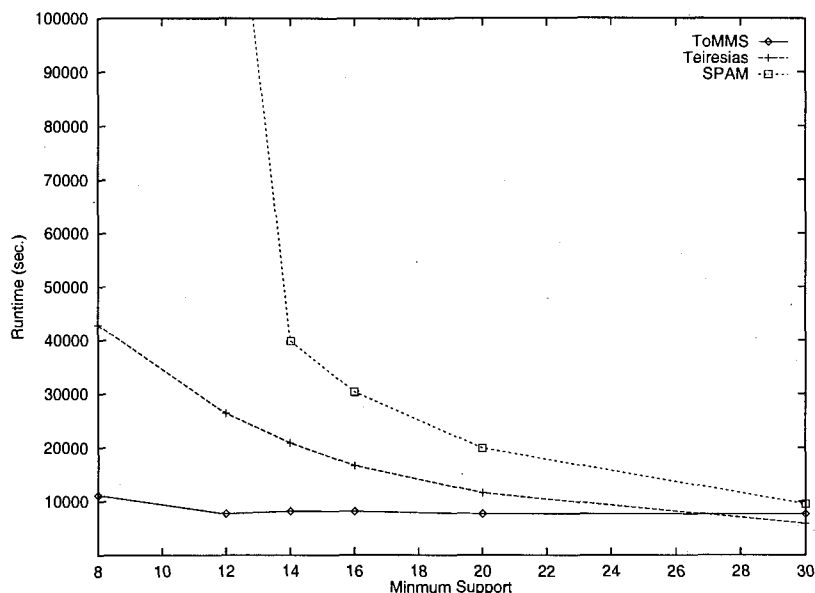


Figure 5.1: Runtime on outer membrane protein sequence dataset.

We obtained similar results for other concept graphs [18] [24]. Protein sequence datasets are in general much denser than transactional datasets which implies that the frequent patterns in protein sequence datasets usually are much longer than the frequent patterns in transactional datasets. The density of the three test datasets also varies significantly. For example, at a minimum support of 20, the outer membrane protein sequence dataset has 599,307 frequent patterns, the non outer membrane protein sequence dataset has 503,895 frequent patterns and the yeast protein sequence dataset has 123,002 frequent patterns.

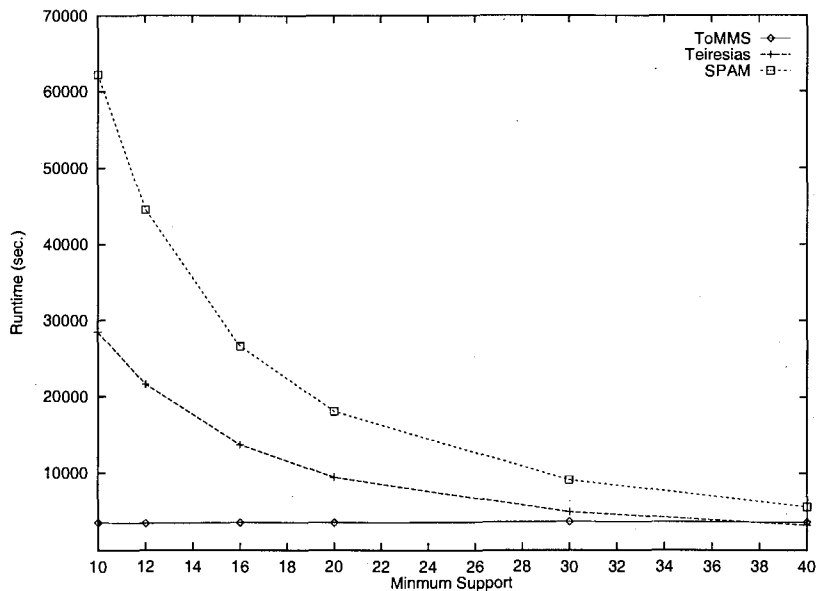


Figure 5.2: Runtime on non outer membrane protein sequence dataset.

On all three datasets, ToMMS clearly outperformed SPAM and Teiresias up to minimum support values of around 10%. Note that the minimum support values must be in this relatively low range in order to discover frequent patterns that are long enough to be biologically meaningful. E.g., in the outer membrane protein sequence dataset, the length of the longest frequent patterns is 33 at a minimum support of 8, but only 12 at a minimum support of 16 and only 9 at a minimum support of 40. For the prediction of the localization of outer membrane proteins [22], beta-strand patterns are crucial which have a length of up to 20 amino-acids.

We conclude that ToMMS is the method of choice for low minimum support values, whereas Teiresias is the most efficient method for high minimum support values.

The runtime of both competitors increases exponentially with decreasing minimum support. On the other hand, ToMMS, as shown in the results, exhibits a nearly constant runtime. This is due to the following reason. When the minimum support decreases, the generation of a single frequent pattern will require a smaller number of generalizations and, thus, will become cheaper. But, on the other hand, when minimum support decreases, more patterns will be frequent and will be discovered which is shown in figure 11. Because of this trade-off, ToMMS shows a nearly constant runtime. This is a very nice property since it allows us to

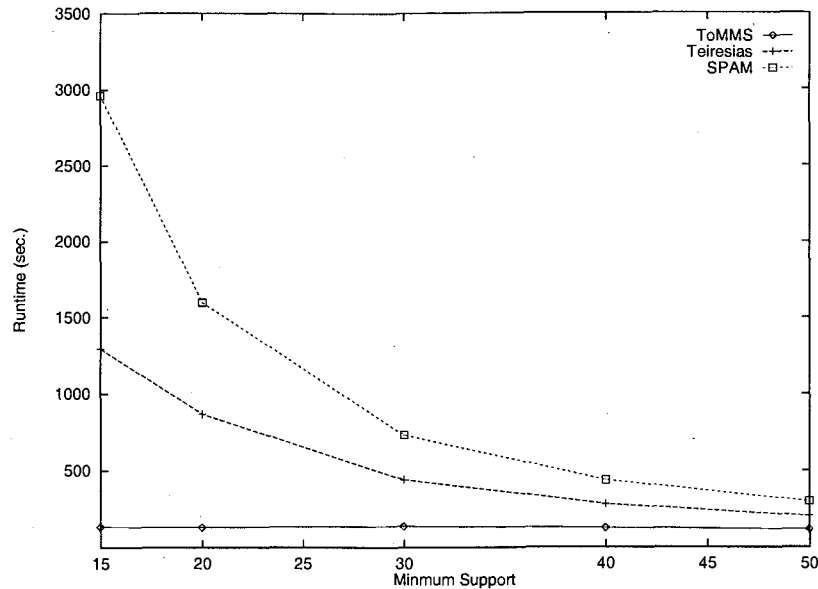


Figure 5.3: Runtime on yeast protein sequence dataset.

reliably estimate the runtime of ToMMS.

Figure 5.4 shows the runtime for determining the maximal length and the total runtime of ToMMS, again on the yeast sequence dataset. As we can see from this figure, the overhead of determining the maximal length before starting the top-down pattern enumeration is relatively small. This conforms the statement in section 4.5. Determining the maximal length of frequent patterns before enumerating all of them provides a powerful tool. The traditional bottom-up approach can not find out the maximal until the whole mining process ends. How to set proper minimum support to get patterns of expected length can not be solved efficiently by this approach. On one hand, at high minimum support, the longest pattern can still be too short to have any biological meaning. Using the bottom-up approach will have to performing the whole experiment again from the beginning at some lower minimum support to get longer patterns. On the other hand, the minimum support can not set to be too low, since this will result a huge number of patterns and the run time of the experiment can be unpredictably long. For example, the experiment on outer membrane protein sequence data, at minimum support 12, the whole mining procedure using SPAM is around 40 hours. By using bottom-up search strategy, ToMMS can efficiently discover the maximal length of patterns. Thus we can use it to get the proper minimum support easily.

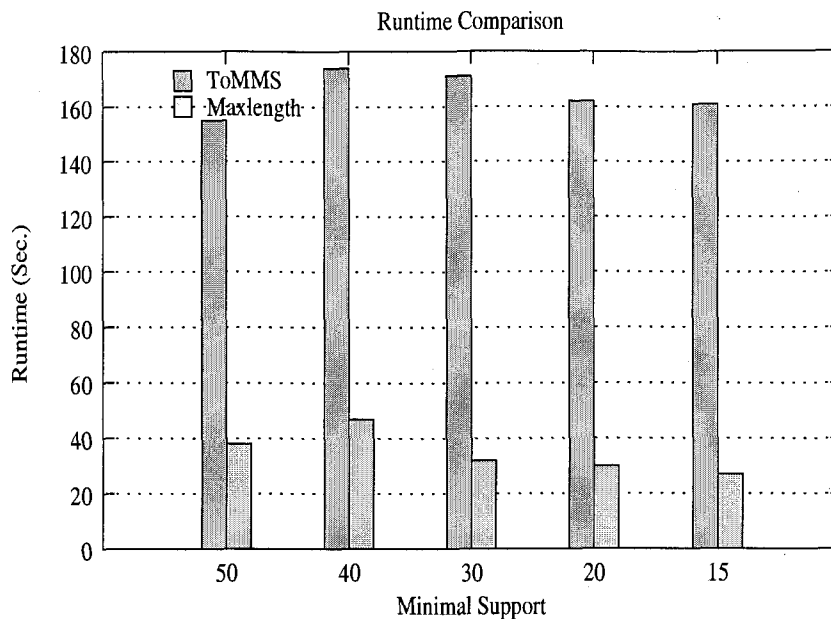


Figure 5.4: Runtime of finding maximal length compare to runtime of ToMMS.

Figure 5.5 shows the scalability of ToMMS with respect to the number of data sequences. Each time we randomly select a subset of the yeast sequence dataset. The result shows that ToMMS has a very good scalability.

Figure 5.6 shows the number of patterns discovered by ToMMS, Teiresias and SPAM on the yeast sequence dataset. SPAM discovers the set of all frequent patterns. Teiresias finds the set of closed patterns where the definition of closed pattern is based on the occurrence, not the support. The number of patterns reported by Teiresias is quite close to the number of all frequent patterns, ToMMS, however significantly reduces the number of output patterns. This is a very desirable effect in most applications of frequent sequence mining, in particular if the patterns shall be analyzed by a domain expert. The set of most specific frequent patterns provides a compact representation of the set of all frequent patterns.

In summary, ToMMS is the most efficient method at low minimum support values. Determining the maximal length of frequent patterns before enumerating all of them provides a powerful tool to set the proper minimum support. And ToMMS scales very well on datasets of different size.

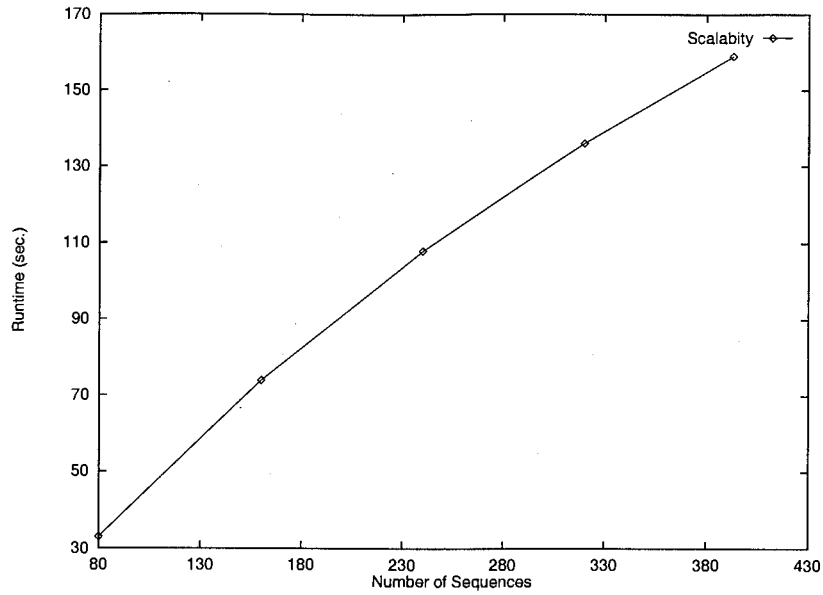


Figure 5.5: Scalability of ToMMS.

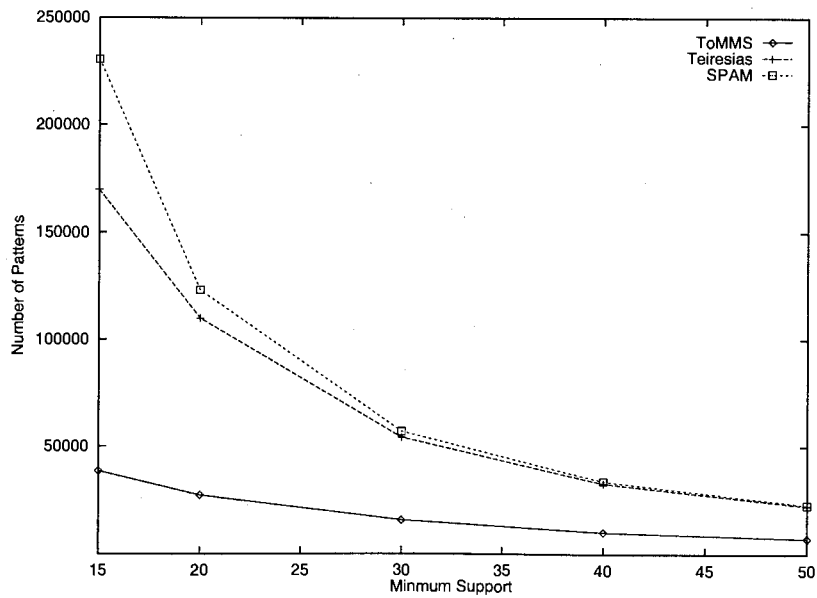


Figure 5.6: Number of patterns discovered by ToMMS, Teiresias and SPAM.

Chapter 6

Conclusion

The explosive growth in the amount of biological information necessitates the use of computers to make sense of this flood of data. One promising approach for mining biological sequence data is mining frequent patterns. In this thesis, we addressed the problem of mining most specific frequent patterns in biological sequence data in the presence of concept graphs, and proposed an efficient algorithm which adopts top-down approach for mining such patterns. In this chapter, we summarize the thesis and discuss some future research directions.

6.1 Summary of Thesis

We summarized our contributions in this thesis as follow:

- We introduced a novel approach for mining most specific frequent patterns in biological sequence data, which performing top-down pattern enumeration. Infrequent patterns are generalized until they become frequent by either reducing a pattern to a subsequence or by replacing some of its elements by predecessors in the concept graph. This approach seems to be more appropriate than the classic bottom-up approach for mining most specific (not all) patterns if these patterns are long and if we can estimate the maximal length of frequent patterns.
- The proposed ToMMS method includes an algorithm to efficiently determine the exact value of this maximal length without enumerating all of the frequent patterns. This is very helpful in setting proper minimum support to get patterns of ideal length.

Traditional bottom-up approach will have to explore the whole search space before determining the maximal length.

- By shortening and pairwise generalizing infrequent patterns, ToMMS will never enumerate a potential pattern which does not exist in the database, i.e. the enumerated patterns at least have one supporting sequence.
- A thorough experimental evaluation on real life biological sequence datasets demonstrated that ToMMS clearly outperforms the state-of-the-art methods from the KDD as well as from the bioinformatics community for reasonably low minimum support values. Different from the bottom-up methods, the runtime of ToMMS is very robust to the minimum support value which allows us to reliably estimate the runtime of ToMMS before starting potentially very expensive experiments.

6.2 Future Research Directions

Future work includes the following issues:

- First, we want to investigate the impact of different types and numbers of sequential patterns in different data mining applications. Frequent pattern-based data mining methods have been very successful, for instance, for the functional classification of biological sequences [22] or the clustering of large sets of proteins [9]. However, existing work does not address what kind of difference will occur by using different types and numbers of patterns.
- Integrate the ultimate goal of KDD (e.g. clustering and classification) into the process of mining frequent patterns. Although frequent pattern mining has been extensively studied during past years, the usefulness of those patterns has not received so much attention. Usually the process of mining frequent patterns and the process of using the patterns are independent. How to integrate these two parts is one interesting direction we would like to explore.
- In this thesis, we have focused on biological sequence data. However, the top-down pattern enumeration approach is promising whenever data sequences, and in particular, frequent patterns are long. Therefore, other applications of this kind such as databases of text documents (a document is a sequence of terms) and web logs (a user

session is a sequence of actions) should be explored to get a better understanding of the merits of both, the top-down and bottom-up pattern enumeration paradigm.

Bibliography

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD Conference*, pages 207–216, 1993.
- [2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering*, pages 3–14, 1995.
- [3] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick. Sequential pattern mining using a bitmap representation. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.
- [4] DA. Benson, I. Karsch-Mizrachi, DJ. Lipman, J. Ostell, BA. Rapp, and DL. Wheeler. Genbank. *Nucleic Acids Research*, 30:17–20, 2002.
- [5] B. Boeckmann, A. Bairoch, R. Apweiler, MC. Blatter, A. Estreicher, E. Gasteiger, MJ. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic Acids Research*, 31:365–370, 2003.
- [6] C. Branden and J. Tooze. *Introduction to protein structure*. New York & London: Garland Publishing, 1991.
- [7] A. Brazma, L. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5:279–305, 1998.
- [8] A. Floratos. Pattern discovery in biology: Theory and applications. *Ph.D Dissertation*, 1999.
- [9] V. Guralnik and G. Karypis. A scalable algorithm for clustering sequential data. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, 2001.
- [10] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and MC. Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.

- [11] S. Henikoff and JG. Henikoff. Amino acid substitution matrices from protein blocks. In *Proceedings of the National Academy of Science(USA)*, volume 89, pages 10915–10919, 1989.
- [12] B. Lewin. *Genes VI*. Oxford University Press, 1997.
- [13] D. Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, pages 322–336, 1978.
- [14] F. Masegila, F. Cathala, and P. Poncelet. The psp approach for mining sequential patterns. In *Proceedings of the 1998 European Symposium on Principle of Data Mining and Knowledge Discovery*, pages 176–184, 1998.
- [15] TM. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Fifth International Joint Conference on AI*, pages 305–310. Cambridge, MA: MIT Press, 1977.
- [16] TM. Mitchell. *Version Space: An approach to concept learning*. Electrical Engineering Dept., Standford University, 1979.
- [17] TM. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
- [18] G. Mocz. Fuzzy cluster analysis of simple physicochemical properties of amino acids for recognizing secondary structure in proteins. *Protein Science*, 4:1178–1187, 1995.
- [19] CG. Nevill-Manning, KS. Sethi, TD. Wu, and DL. Brutlag. Enumerating and ranking discrete motifs. In *Proceedings of the 5th International Conference on Intellegent Systems for Molecular Biology*, pages 202–209, 1997.
- [20] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and MC. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering*, pages 215–224, 2001.
- [21] I. Rigoutsos and A. Floratos. Combinatorial pattern discovery in biological sequences: the teiresias algorithm. *Bioinformatics*, 14:55–67, 1998.
- [22] R. She, F. Chen, K. Wang, M. Ester, JL. Gardy, and FSL Brinkman. Frequent-subsequence-based prediction of outer membrane proteins. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.
- [23] H.O. Smith, T.M. Annau, and S. Chandrasegaran. Finding sequence motifs in groups of functionally related proteins. In *Proceedings of the National Academy of Science(USA)*, volume 87, pages 826–830, 1990.

- [24] RF. Smith and TF. Smith. Automatic generation of primary sequence patterns from sets of related proteins. In *Proceedings of the National Academy of Science(USA)*, volume 87, pages 118–122, 1990.
- [25] R. Srikant and R. Agrwal. Mining sequential patterns: Generalizatin and performance improvements. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 3–17, 1996.
- [26] WR. Taylor. The classification of amino-acid conservation. *Journal of Theoretical Biology*, 19:205–218, 1986.
- [27] I. Tsoukatos and D. Gunopulos. Efficient mining of spatiotemporal patterns. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases*, pages 425–442, 2001.
- [28] JD. Watson, NH. Hopkins, JW. Roberts, J. Steitz, and AM. Weiner. *Molecular Biology of the Gene*. The Benjamin/Cummings Publishing Company, 1987.
- [29] TD. Wu and DL. Brutlag. Discovering empirically conserved amino acid substitution groups in databases of protein families. In *Proceedings of the 4th International Conference on Intelligent Systems for Molecular Biology*, pages 230–240, 1996.
- [30] MJ. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42:31–60, 2001.