

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

Martin Ester  
Simon Fraser University  
School of Computing Science  
Burnaby BC, Canada V5A 1S6  
++1-604-291 4411  
ester@cs.sfu.ca

Xiang Zhang  
Simon Fraser University  
School of Computing Science  
Burnaby BC, Canada V5A 1S6  
++1-604-291 5371  
xzhangj@cs.sfu.ca

## 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.

## Keywords

Data mining algorithms, sequence data, efficiency, data mining applications in bioinformatics.

## 1. INTRODUCTION

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 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. Obviously, manual analysis of the large databases is infeasible, and data mining is the direction of choice. 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 [18], PrefixSpan [13], SPADE[21], DFS-MINE[19], SPAM[2], CloSpan [20]). 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 propose the following approach:

- We restrict the result of pattern mining to the 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 [14] has been proposed which takes a similar approach. It determines the set of all closed frequent patterns and also uses concept graphs to discover generalised patterns. Like the methods from the KDD community, Teiresias adopts the paradigm of bottom-up pattern enumeration.

In this paper, we introduce a novel method, ToMMS, 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. 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. 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 not too high minimum support values. Such minimum support values are necessary to discover frequent patterns long enough to be biologically meaningful. Extensive empirical studies [15] showed that state-of-the-art classification methods perform best when the input frequent patterns have low minimum support.

ToMMS has the following properties distinguishing it from existing algorithms. 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 candidate pattern which does not exist in the database, i.e. enumerated patterns have at least one supporting sequence. This will greatly reduce the search space. Finally, ToMMS can efficiently determine the maximal length of all frequent patterns before enumerating all of them.

The rest of this paper is organized as follows. Section 2 surveys related work. In section 3, we introduce our notion of most specific frequent patterns and present ToMMS, a new top-down method for mining such patterns. Concept graphs deserve special attention because they blow up the search space, and their efficient treatment is discussed in section 4. In section 5, we report the results of an experimental evaluation and comparison with state-of-the-art methods. Section 6 summarizes our contributions and outlines directions for future research.

## 2. RELATED WORK

The topic of mining frequent sequential patterns has received a lot of attention in the KDD community. In the bioinformatics community, a lot of research on mining patterns in biological sequence data has been conducted. In this section, we briefly survey representative methods from both of these areas with respect to the following criteria:

- 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 concept graph?

GSP [18] is one of the first methods proposed in the KDD community. It finds all sequential patterns in a bottom-up, breadth-first manner. Concept graphs are incorporated by simply transforming a sequence when counting its support. PrefixSpan [13] determines all frequent patterns based on the FP-tree which induces a bottom-up, depth-first search strategy. PrefixSpan is much more efficient than GSP but does not support concept graphs, and there is no straightforward way of incorporating them. SPADE [21] also finds all frequent sequential patterns adopting a bottom-up, depth-first strategy. A vertical representation of the dataset is used which for each pattern stores a list of its occurrences in the data sequences. The vertical representation of the dataset allows very efficient support

counting. DFS-MINE [19] reports only maximal frequent sequential patterns and employs a bottom-up, depth-first search strategy. A list of maximal frequent patterns, a list of minimal non frequent patterns and the set of items that a frequent pattern must not be extended with to generate longer candidate patterns are used to prune the search space. DFS-MINE does not address the problem of incorporating concept graphs. By exploring global optimization techniques, CloSpan [20] mines closed frequent sequential patterns. However, it also does not support concept graphs. The recently proposed SPAM [2] method is similar to SPADE. The major difference is that bitmaps are used instead of linked 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.

In the KDD community, maximal frequent itemsets mining (Max-Miner [3], MAFIA [7] and GenMax [8]) is also related to our work. Max-Miner employs a breadth-first traversal of the search space enhanced with a look-ahead pruning strategy: if for a node in the search tree all its extensions are determined to be frequent, there is no need to further process that node. MAFIA and GenMax integrate depth-first traversal of the search space with effective pruning mechanisms. However, all these methods are not designed to mine sequential patterns and their basic search strategy is bottom-up.

In the bioinformatics community, a wealth of methods for determining frequent sequential patterns has been developed. [6] presents a good overview on this research. Teiresias [14] 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. While the original proposal did not involve concept graphs, it has later been extended to support this functionality. 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. Taking a totally different approach, multiple alignment based methods for discovering so-called motifs have become popular in the bioinformatics community [10]. In a top-down manner, the input sequences are pairwise aligned until they become frequent. Concept graphs are implicitly incorporated in the similarity matrix for pairs of amino acids which is exploited by the sequence alignment algorithm. A drawback of this approach is that the problem of determining an optimal alignment is NP-hard such that heuristic algorithms have to be used which

produce only suboptimal solutions. Furthermore, these methods return only a very small, not precisely defined subset of the set of all frequent sequential patterns and can, therefore, hardly be compared with the other methods discussed above.

### 3. MINING MOST SPECIFIC FREQUENT PATTERNS

#### 3.1 The Problem of Mining Most Specific Frequent Patterns

In this subsection, we introduce the problem of mining most specific frequent patterns.

Let  $\Sigma$  be a set of symbols, the *alphabet*. The database  $D$  consists of  $n$  *database sequences*  $s = s_1s_2 \dots s_l$  with  $s_i \in \Sigma$  for  $1 \leq i \leq l$ .  $l$  is the *length* of  $s$ . For protein sequence databases, e.g.,  $\Sigma = \{K, E, Q, D, N, P, R, S, T, G, A, Y, H, I, W, L, M, V, F, C\}$ , i.e. the one-letter codes of the 20 amino acids.

Let  $\Omega$  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 paper, we use small letters to denote the concepts. For example,  $s$  may represent the property *small*,  $p$  the property *polar*.

Based on  $\Sigma$  and  $\Omega$ , we define a *concept graph*  $\Gamma$  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  $X$ . The *distance* of two vertices  $v_i$  and  $v_j$  in  $\Gamma$ , denoted by  $dist(v_i, v_j)$ , is the number of the edges of the shortest path between  $v_i$  and  $v_j$ . Figure 1 shows a simple concept graph. We will use it as the running example in this paper.

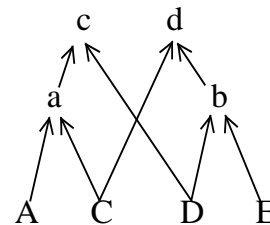


Figure 1: Example of a concept graph

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

$\forall X \in \Sigma : level(X) = 0;$

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

A concept graph  $\Gamma$  induces a partial order *more specific*, denoted by  $<_{\Gamma}$ , on  $\Sigma \cup \Omega$  as follows:

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

The order  $<_{\Gamma}$  has the following properties:

$\forall e, f \in \Sigma \cup \Omega : e <_{\Gamma} f \Rightarrow \neg(f <_{\Gamma} e)$  (antisymmetric);

$\forall e, f, g \in \Sigma \cup \Omega : (e <_{\Gamma} f) \wedge (f <_{\Gamma} g) \Rightarrow (e <_{\Gamma} g)$  (transitive).

We can extend  $<_{\Gamma}$  to  $\leq_{\Gamma}$  which is transitive and reflexive:

$\forall e \in \Sigma \cup \Omega : e \leq_{\Gamma} e$  (*reflexive*). Intuitively, if  $X \leq_{\Gamma} Y$ , then  $Y$  subsumes  $X$ , i.e.  $Y$  is either identical to  $X$  or  $Y$  generalizes  $X$ .

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

(1)  $(X \leq_{\Gamma} Z) \wedge (Y \leq_{\Gamma} Z),$

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

In figure 1, for example,  $level(A)=0$ ,  $level(d)=2$ ,  $A <_{\Gamma} a$ ,  $sp(A, C) = a$ .

A *pattern* is a sequence  $p = e_1 e_2 \dots e_l$  with  $e_i \in \Sigma \cup \Omega$ , and  $l$  is the *length* of the pattern, denoted by  $length(p)$ . A pattern  $p = e_1 e_2 \dots e_j$  is a *subsequence* of a pattern  $p' = e'_1 e'_2 \dots e'_i$  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  $\Sigma$ , for example  $CDE$ . A non-concrete pattern such as  $CbE$  is called a *generalized pattern*.

We extend the definition of the level and the relation  $<_{\Gamma}$  from single symbols to whole patterns as follows. The *level* of a pattern w.r.t. graph  $\Gamma$  is the sum of the levels of all elements of the pattern in the graph  $\Gamma$ , i.e. the *level* of a pattern  $p = e_1 e_2 \dots e_l$  is  $level(p) = \sum_{i=1}^l level(e_i)$ . 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  $p' = e'_1 e'_2 \dots e'_i$  is called to be *more specific* than a pattern  $p = e_1 e_2 \dots e_j$  ( $i \geq j$ ) 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, using the concept graph shown in figure 1, pattern  $CDE$  has level 0, pattern  $cbE$  has level 3 and

$aED <_{\Gamma} abD$ . Note that a pattern is not more specific than itself. We again extend  $<_{\Gamma}$  to  $\leq_{\Gamma}$  by defining: for all  $p$  in  $\Sigma \cup \Omega : p \leq_{\Gamma} p$ . By definition,  $\leq_{\Gamma}$  defined on patterns is also a partial order.

A database sequence  $s$  *supports* a pattern  $p$  if  $s$  is more specific than  $p$ , i.e.  $s \leq_{\Gamma} p$ . Sequence  $s$  is called a *supporting sequence* of  $p$ . A subsequence  $p'$  of  $s$  with (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  $\delta$ , 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  $\delta$  and concept graph  $\Gamma$  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 proposition shows that the set of all frequent patterns can be derived from the set of all most specific frequent patterns by generalizing one of them.

### Proposition 1

Let  $S$  be the set of all most specific frequent patterns in database  $D$  w.r.t.  $\delta$  and concept graph  $\Gamma$ .

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  $\exists p_1 : (sup(p_1) \geq \delta) \wedge (p_1 <_{\Gamma} p)$  because then  $p$  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.  $\delta$  (because a frequent pattern generalizes all its supporting database sequences and, therefore, cannot be longer than these sequences). The case (1) must eventually apply to

$q = p$  and we obtain a pattern  $q \in S$  with  $q = p_n \leq_r \dots \leq_r p_2 \leq_r p_1 \leq_r p$ . Because of the transitivity of  $\leq_r$ , we conclude  $q \leq_r p$ . ■

### 3.2 ToMMS: A Top-Down Method for Mining Most Specific Frequent Patterns

Given a sequence database  $D$ , a concept graph  $\Gamma$  and a minimum support value  $\delta$ , 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 2 shows the pseudo-code of ToMMS using the notations introduced in Table 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 1: Notations**

**ToMMS** ( database  $D$ , min\_sup  $\delta$ , concept graph  $\Gamma$  )

max-length  $\leftarrow$  Maxlength (  $D$ ,  $\delta$ ,  $\Gamma$  );

$i =$  max-length;

$CP_i \leftarrow$  Preprocess (  $D$ , max-length );

**while**  $CP_i \neq \emptyset$  **do**

$MSFP_i \leftarrow \{ c \mid c \in CP_i, c \text{ is frequent} \};$

$FCP_i \leftarrow FCP_i \cup \{ c \mid c \in CP_i, c \text{ is frequent} \};$

$ICP_i \leftarrow \{ c \mid c \in CP_i, c \text{ is infrequent} \};$

$MSFP_i \leftarrow MSFP_i \cup \text{Generalize} ( ICP_i, FCP_i );$

$MSFP_i \leftarrow \text{Subseqcheck} ( MSFP_i );$

$CP_{i-1} \leftarrow \{ \text{Shorten} ( ICP_i ) \};$

$FCP_{i-1} \leftarrow \{ \text{Shorten} ( FCP_i ) \};$

$i \leftarrow i - 1;$

**end-while** ;

**return**  $\bigcup_i MSFP_i$  ;

**Figure 2: Pseudo-code of ToMMS**

Let us assume that max-length is 4. ToMMS starts from the only subsequence of length 4, e.g. the data sequence  $ACDE$ , 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  $ACD$  and  $CDE$ . If any one of them is frequent, we output it and stop generating its subsequences. For example, if  $ACD$  turns out to be frequent, we do not check the subsequences  $AC$  and  $CD$ . Suppose the maximum length of the database sequences is  $l$  and the number of these sequences 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.

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, 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  $ADC$  and  $AEC$  can be minimally generalized to  $AbC$ . 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.

The function Shorten shortens all patterns contained in the input set by 1 element. For example, the set  $\{ACD, CDE\}$  is shortened to the set  $\{AC, CD, DE\}$ .

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).

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 3.3.

### 3.3 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  $max-length = current\ length$  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 of length  $l$  is frequent, then very few generalized patterns will be generated by matching infrequent concrete patterns. (See section 4.1 for the introduction of the match operation.) 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. INCORPORATING CONCEPT GRAPHS

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 and show the correctness of the ToMMS algorithm.

### 4.1 Search strategy for generalized patterns

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 top-down search, i.e. generalize only infrequent patterns 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)  $\forall p' \neq p: (p_1 \leq_{\Gamma} p') \wedge (p_2 \leq_{\Gamma} p') \wedge \dots \wedge (p_n \leq_{\Gamma} p') \Rightarrow \neg(p' <_{\Gamma} p)$ .

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 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)  $(p_1 \leq_{\Gamma} p') \wedge (p_2 \leq_{\Gamma} p') \wedge \dots \wedge (p_n \leq_{\Gamma} p')$ ,

there is a pattern  $p \in S$  such that  $p \leq_{\Gamma} p'$ .

Proof: Similar to the proof of Proposition 1. ■

Theorem 2 shows that any match of more than two patterns is equivalent to a sequence of pair-wise matches of the same patterns.

### Theorem 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  $\forall p_m \in \oplus(p_1, p_2, \dots, p_{n-1}), p \notin \oplus(p_m, p_n)$ . 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  $p \in \oplus(p_1, p_2, \dots, p_n)$  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 Naïve 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 level 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 1

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:

$$\text{level}(p_1) = \text{level}(p_2) \Rightarrow \neg(p_1 <_{\Gamma} p_2) \wedge \neg(p_2 <_{\Gamma} p_1).$$

### Lemma 2

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:  $\text{level}(p_1) < \text{level}(p_2) \Rightarrow \neg(p_1 >_{\Gamma} p_2)$ .

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 1, we do not have to test for  $<_{\Gamma}$  relationships among patterns of the same level (the = relationship is still possible). According to lemma 2, 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.

Figure 3 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  $i$ .

```

Generalize (ICPi, FCPi )
for j from 1 to max-level do
  GPij ← { p | p = pm ⊕ pn, pm, pn ∈ ICPi, j = level (p)};
end-for
for j = 1 to max-level do
  for each p ∈ FCPi do
    mark p' where p' ∈ GPij, p' ≥Γ p, j = level (p);
  end-for;
  IGPij ← { p | p ∈ GPij, p is infrequent and unmarked };
  for each pair (pa, pb), pa ∈ ICPi, pb ∈ IGPij do
    if (SupSeq(pa) ⊂ SupSeq(pb) ∧ SupSeq(pb) ⊂ SupSeq(pa))
      GPil ← GPil ∪ { p | p = pa ⊕ pb, l = level (p) };
    end-for;
  FGPij ← { p | p ∈ GPij, p is frequent };
  for each p ∈ FGPij do
    mark p' where p' ∈ GPil, p' ≥Γ p, l = level (p);
  end-for
end-for
return ∪j FGPij

```

**Figure 3: Pseudo-code of Generalization**

Let us use again the concept graph in figure 1. Suppose current length  $i$  is 3, minimum support is 3 and the set of ICP<sub>3</sub> is as shown in figure 4 (a). The ids of the supporting sequences are also given, for example, the supporting sequence of  $ACD$  is sequence 1. After pairwise generalizing all infrequent concrete pattern, the resulting sets GP<sub>3i</sub> are shown in figure 4 (b). Starting from the lowest level nonempty generalised pattern set GP<sub>32</sub>, we match the infrequent patterns in GP<sub>32</sub> with the patterns in ICP<sub>3</sub>. The result of this generalization is shown in figure 4 (c). All the possible 3 matches (where the sets of supporting sequences are not subsets of one another) of one pattern in ICP<sub>3</sub> and one pattern in GP<sub>32</sub> yield the same result aab.

- (a) ICP<sub>3</sub>={ $ACD$ : [1],  $CAD$ : [2],  $CCE$ : [3]}
- (b) GP<sub>31</sub>=∅  
 GP<sub>32</sub>={ $aaD$ : [1,2],  $aCb$ : [1,3],  $Cab$ : [2,3]}  
 GP<sub>33</sub>=∅
- (c) GP<sub>31</sub>=∅  
 GP<sub>32</sub>={ $aaD$ : [1,2],  $aCb$ : [1,3],  $Cab$ : [2,3]}  
 GP<sub>33</sub>={ $aab$ : [1,2,3]}

**Figure 4: Example of function Generalize**

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.

## 4.2 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 3) and only most specific frequent patterns are reported (theorem 4).

### Theorem 3

ToMMS reports all most specific frequent patterns in database  $D$  w.r.t. concept graph  $\Gamma$  and minimum support  $\delta$ .

Proof:

1) Let  $p$  be a *concrete* most specific frequent patterns in database  $D$  w.r.t. concept graph  $\Gamma$  and minimum support  $\delta$ . 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,  $p \in S$  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  $\Gamma$  and minimum support  $\delta$ . 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  $p$ . From theorem 1 we know that there is a  $q \in \oplus(p_1, p_2, \dots, p_m)$  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.  $p \in \oplus(p_1, p_2, \dots, p_m)$ . According to theorem 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  $p_1, p_2, \dots, p_m$ . (i) Furthermore,  $\forall i, 1 \leq i \leq m$ :  $p_i$  is infrequent because of the following reason. Since all  $p_i$  are concrete patterns,  $\forall 1 \leq i \leq m$ :  $p_i <_{\Gamma} p$ , i.e.  $\forall 1 \leq i \leq m$ :  $\sup(p_i) < \sup(p)$ .  $p$  is most specific, and, therefore all  $p_i$  are infrequent. (ii) Because of (i) and (ii),  $\oplus(p_1, p_2, \dots, p_m)$  will be constructed and this generates  $p$ . ■

#### Theorem 4

All patterns reported by ToMMS are most specific frequent patterns in database  $D$  w.r.t. concept graph  $\Gamma$  and minimum support  $\delta$ .

Proof: Let  $p$  be a pattern reported by ToMMS.

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  $p$ .

(i)  $length(q) < length(p)$

ToMMS generates patterns in descending order of length.

(ii)  $length(q) = length(p)$

ToMMS generates same length patterns in ascending order of level. According to lemmas 1 and 2, 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. ■

## 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 [15]. 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](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 below.

Data	number of Sequences	Min. Length	Max. Length	Ave. Length
OM	417	91	3705	579
non OM	620	68	1553	392
Yeast	393	15	1859	256

**Table 2. Description of the evaluation datasets**

We use a typical concept graph of amino-acids as used in [12]. The table below shows the sets of amino acids corresponding to the concepts of this concept graph. These concepts represent the relevant physico-chemical properties of amino-acids.

Concepts (Physico-chemical properties)	Amino acids
Small	AG
Small hydroxyl	ST
Basic	KR
Aromatics	FWY
Basic	HKR
Small hydrophobic	ILV
Medium hydrophobic	ILMV
Acidic/amid	EDNQ
Small polar	AGPST

**Table 3. Concepts of amino acids**

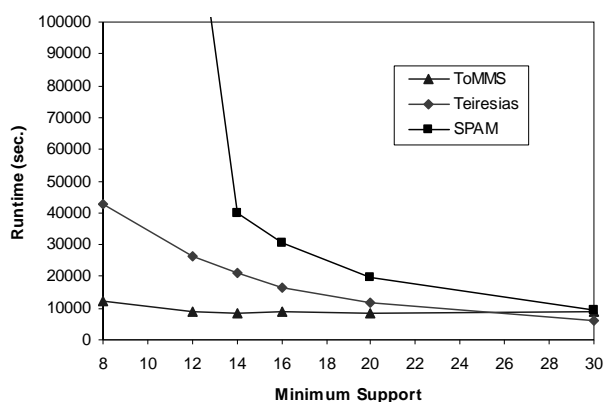
### 5.2 Design of the Experiments

We chose the competitors of ToMMS as follows. SPAM [2] has been shown to outperform all other algorithms for sequential pattern mining in the data mining community. We implemented SPAM in a modified version extending the alphabet by concepts (to incorporate concept graphs) and generating patterns without gaps (to make it comparable to 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 [14] 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. We configured Teiresias to find the same type of patterns as ToMMS, i.e. patterns incorporating concepts but without gaps. All programs are written in Microsoft Visual C++ 6.0. The experiments were performed on a 1.7 GHz

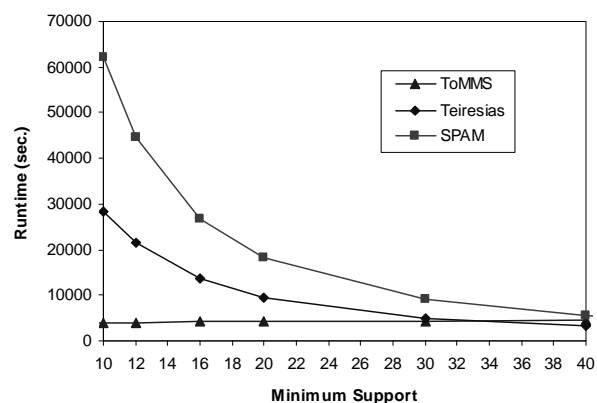
Pentium PC machine with 1 Gigabytes main memory, running Microsoft Windows 2000.

### 5.3 Experimental Results

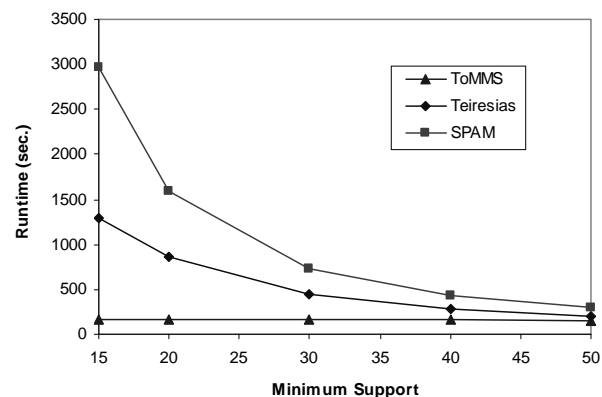
Figures 5 to 7 show the runtimes of ToMMS, Teiresias and SPAM on the three test datasets at different (absolute) minimum support values using the concept graph from [12]. We obtained similar results for other concept graphs [11] [17]. Protein sequence datasets are in general much denser than transactional 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. 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 [15], beta-strand patterns are crucial which have a length of up to 20 amino acids. Experimental studies in [15] also showed that classification tools taking frequent patterns as input perform best when the input patterns have very low minimum support.



**Figure 5. Runtime on outer membrane protein sequence dataset**



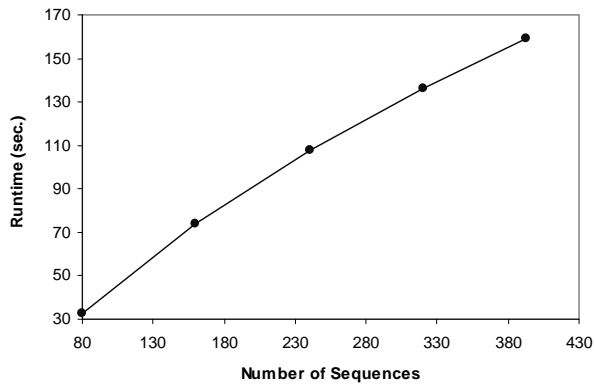
**Figure 6. Runtime on non outer membrane protein sequence dataset**



**Figure 7. Runtime on yeast protein sequence dataset**

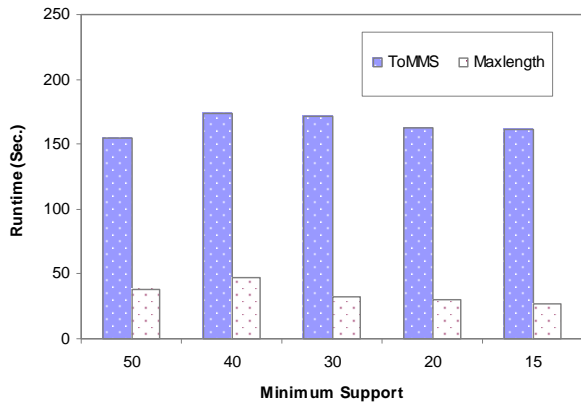
We conclude that ToMMS is the method of choice for low minimum support values, whereas Teiresias is the most efficient method for high minimum support values.

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 10. Because of this trade-off, ToMMS shows a nearly constant runtime. This is a very nice property since it allows us to reliably estimate the runtime of ToMMS.



**Figure 8. Scalability of ToMMS**

Figure 8 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.

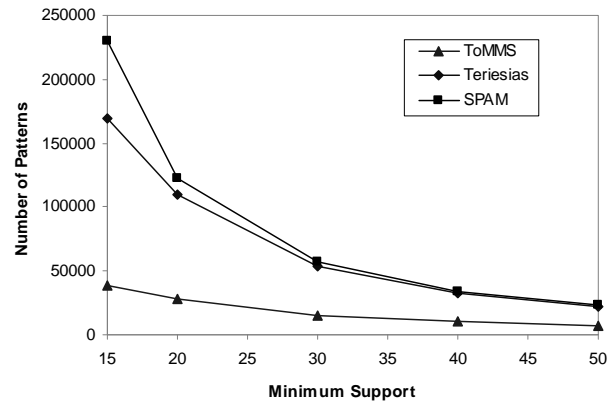


**Figure 9. Runtimes of finding max. length and of ToMMS**

Figure 9 shows the runtime for determining the maximal length and the total runtime of ToMMS, again on the yeast sequence dataset. It turns out that the overhead of determining the maximal length before starting the top-down pattern enumeration is relatively small.

Figure 10 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.



**Figure 10. Number of patterns discovered**

## 6. CONCLUSIONS

In this paper, we addressed the problem of mining most specific frequent patterns in biological sequence data in the presence of concept graphs. Using concept graphs, much longer frequent patterns can be discovered which are more meaningful from a biological point of view. We introduced a novel approach for mining such patterns 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. 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. Teiresias performed best for high values of the minimum support threshold. 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.

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 [15] or the clustering of large sets of proteins [9]. Second, in this paper, 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.

## 7. REFERENCES

- [1] R. Agrawal and R. Srikant. 1995. Mining sequential patterns. *Proceedings of the 11th International Conference on Data Engineering*, 3-14.
- [2] J. Ayres, J. E. Gehrke, T. Yiu, and J. Flannick. 2002. Sequential pattern mining using a bitmap representation. *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 429-435.
- [3] R. J. Bayardo. 1998. Efficiently mining long patterns from databases. *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, 85-93.
- [4] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler. 2002. GenBank. *Nucleic Acids Research*, 30 (1): 17-20.
- [5] B. Boeckmann, A. Bairoch, R. Apweiler, M. C. Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilboud, M. Schneider. 2003. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31 (1): 365-370.
- [6] A. Brazma, L. Jonassen, I. Eidhammer, and D. Gilbert. 1998. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5 (2): 279-305.
- [7] D. Burdick, M. Calimlim and J. Gehrke. 2001. MAFIA: a maximal frequent itemset algorithm for transactional databases. *Proceedings of the 17th International Conference on Data Engineering*, 443-452.
- [8] K. Gouda and M. J. Zaki. 2001. Efficiently mining maximal frequent items. *Proceedings of the 2001 IEEE International Conference on Data Mining*, 163-170.
- [9] V. Guralnik, G. Karypis. 2001. A scalable algorithm for clustering sequential data. *Proceedings of the 2001 IEEE International Conference on Data Mining*, 179-186.
- [10] M. Hirotsawa, Y. Totoki, M. Hoshida, and M. Ishikawa. 1995. Comprehensive study on iterative algorithms of multiple sequence alignment. *CABIOS*, 11(1): 13-18.
- [11] G. Mocz. 1995. Fuzzy cluster analysis of simple physicochemical properties of amino acids for recognizing secondary structure in proteins. *Protein Science*, 4 (6): 1178-1187.
- [12] C. G. Nevill-Manning, K. S. Sethi, T. D. Wu and D. L. Brutlag. 1997. Enumerating and ranking discrete motifs. *Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology*, 202-209.
- [13] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M-C. Hsu. 2001. PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. *Proceedings of the 17th International Conference on Data Engineering*, 215-224.
- [14] I. Rigoutsos and A. Floratos. 1998. Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. *Bioinformatics*, 14(1): 55-67.
- [15] R. She, F. Chen, K. Wang, M. Ester, J. L. Gardy, F. S. L. Brinkman. 2003. Frequent-subsequence-based prediction of outer membrane proteins. *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 436-445.
- [16] H.O. Smith, T.M. Annau, and S. Chandrasegaran. 1990. Finding sequence motifs in groups of functionally related proteins. *Proceedings of the National Academy of Science (USA)*, 87 (2): 826-830.
- [17] R. F. Smith and T. F. Smith. 1990. Automatic generation of primary sequence patterns from sets of related protein sequences. *Proceedings of the National Academy of Science (USA)*, 87 (1): 118-122.
- [18] R. Srikant and R. Agrawal. 1996. Mining sequential patterns: generalization and performance improvements. *Proceedings of the 15th International Conference on Extending Database Technology*, 3-17.
- [19] I. Tsoukatos and D. Gunopulos. 2001. Efficient mining of spatiotemporal patterns. *Proceedings of the 7th International Symposium on Spatial and Temporal Databases*, 425-442.
- [20] X. Yan, J. Han, and R. Afshar. 2003. Clospan: mining closed sequential patterns in large databases. *Proceedings of the 2003 SIAM International Conference on Data Mining*, 166-177.
- [21] M. J. Zaki. 2001. SPADE: an efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42 (1/2): 31-60.