VOGUE: A Novel Variable Order-Gap State Machine for Modeling Sequences

Bouchra Bouqata Christopher D. Carothers, Boleslaw K. Szymanski, and Mohammed J. Zaki
{boquab,chrisc,szymansk,zaki}@cs.rpi.edu

***Abstract*— We present VOGUE, a new state machine that combines two separate techniques for modeling complex patterns in sequential data: data mining and data modeling. VOGUE relies on a novel Variable-Gap Sequence miner (VGS), to mine frequent patterns with different lengths and gaps between elements. It then uses these mined sequences to build the state machine. Moreover, we propose two variations of VOGUE: C-VOGUE that tends to decrease even further the state space complexity of VOGUE by pruning frequent sequences that are artifacts of other primary frequent sequences; and K-VOGUE that allows for sequences to form the same frequent pattern even if they do not have an exact match of elements in all the positions. However, the different elements have to share similar characteristics. We apply VOGUE to the task of protein sequence classification on real data from the PROSITE and SCOP protein families. We show that VOGUEs classification sensitivity outperforms that of higher-order Hidden Markov Models and of HMMER, a state-of-the-art method for protein classification, by decreasing the sate space complexity and improving the accuracy and coverage.**

***Index Terms*— VOGUE, Data Mining, Data Modeling, Hidden Markov Models**

I. Introduction

Many real world applications, such as in bioinformatics, web accesses, and text mining, encompass sequential/temporal data with long and short range dependencies. Techniques for analyzing such types of data can be classified in two broad categories: pattern mining and data modeling. Efficient pattern extraction approaches, such as association rules and sequence mining, were proposed, some for temporally ordered sequences [3], [53], [54], [74], [89] and others for more sophisticated patterns [16], [40]. For data modeling, Hidden Markov Models (HMMs) [67] have been widely employed for sequence data modeling ranging from speech recognition, to web prefetching, to web usage analysis, to biological sequence analysis [1], [9], [29], [35], [51], [65], [66], [69], [85].

There are three basic problems to solve while applying HMMs to real world problems:

1) **Evaluation:** Given the observation sequence O and a model λ, how do we efficiently compute $P(O|\lambda)$?
2) **Decoding:** Given the observation sequence O, and the model λ, how do we choose a corresponding state sequence $Q = q_1 q_2 ... q_T$ which is optimal in some meaningful sense? The solution to this problem would explain the data.
3) **Learning:** How do we adjust the model λ parameters to maximize $P(O|\lambda)$?

Of all the three problems, the third one is the most crucial and challenging to solve for most applications of HMMs. Due to the complexity of the problem and the finite number of observations, there is no known analytical method so far for estimating λ to maximize globally $P(O|\lambda)$. Instead, iterative methods that provide a local maxima on $P(O|\lambda)$ can be used such as the Baum-Welch estimation algorithm [13].

HMMs depend on the Markovian property, i.e., the current state i in the sequence depends only on the previous state j, which makes them unsuitable for problems where general patterns may display longer range dependencies. For such problems, higher-order and variable-order HMMs [66], [71], [72] have been proposed, where the order denotes the number of previous states that the current state depends upon. Although higher-order HMMs are often used to model problems that display long range dependency, they suffer from a number of difficulties, namely, *high state-space complexity*, *reduced coverage*, and sometimes even *low prediction accuracy* [22]. The main challenge here is that building higher order HMMs [66] is not easy, since we have to estimate the joint probabilities of the previous m states (in an m-order HMM). Furthermore, not all of the previous m states may be predictive of the current state. Moreover, the training process is extremely expensive and suffers from local optima, due to the use of Baum-Welch algorithm [13], which is an Expectation Maximization (EM) method for training the model.

To address these limitations, we propose, in this thesis, a new approach to temporal/sequential data analysis that combines temporal data mining and data modeling via statistics. We introduce a new state machine methodology called **VOGUE** (**V**ariable **O**rder **G**aps for **U**nstructured **E**lements) to discover and interpret long and short range temporal locality and dependencies in the analyzed data. The first step of our method uses a new sequence mining algorithm, called **V**ariable-**G**ap **S**equence miner (VGS), to mine frequent patterns. The mined patterns could be of different lengths and may contain different gaps between the elements of the mined sequences. The second step of our technique uses the mined variable-gap sequences to build the VOGUE state machine.

A. VOGUE OVERVIEW

Let's consider a simple example to illustrate our main idea. Let S be a sequence over the alphabet $\Sigma = \{A, \cdots, K\}$, with $S = \mathbf{ABA}C\mathbf{B}D\mathbf{A}EF\mathbf{B}GH\mathbf{A}IJK\mathbf{B}$. We can observe that $A \rightarrow B$ is a pattern that repeats frequently (4 times), but with variable length gaps in-between. $B \rightarrow A$ is also frequent (3 times), again with gaps of variable lengths. A single order HMM will fail to capture any patterns since no symbol depends purely on the previous symbol. We could try higher order HMMs, but they will model many irrelevant parts of the input sequence. More importantly, *no fixed-order HMM for $k \geq 1$ can model this sequence*, since none of them detects the variable repeating pattern between A and B (or vice versa). This is easy to see, since for any fixed sliding window of size k, no k-letter word (or k-gram) ever repeats! In contrast our VGS mining algorithm is able to extract both $A \rightarrow B$, and $B \rightarrow A$ as frequent subsequences, and it will also record how many times a given gap length is seen, as well as the frequency of the symbols seen in those gaps. This knowledge of gaps plays a crucial role in VOGUE, and distinguishes it from all previous approaches which either do not consider gaps or allow only fixed gaps. VOGUE models gaps via *gap states* between elements of a sequence. The gap state has a notion of state duration which is executed according to the distribution of length of the gaps and the intervening symbols.

The training and testing of VOGUE consists of three main steps: **(i) Pattern Mining** via the novel Variable-Gap Sequence

Bouqata, Carothers, Szymanski and Zaki are with the Computer Science Department at Rensselaer Polytechnique Institute.

(VGS) mining algorithm. **(ii) Data Modeling** via our novel Variable-Order state machine. **(iii) Interpretation** of new data via a modified Viterbi method [26], called Variable-Gap Viterbi (VG-Viterbi), to model the most probable path through a VOGUE model.

B. Contributions

There are several major contributions of our work. First, VOGUE is the combination of two separate but complementary techniques for modeling and interpreting long range dependencies in sequential data: data mining and data modeling. The use of data mining for creating a state machine results in a model that captures the data reference locality better than a traditional HMM created from the original (noisy) data. In addition, our approach automatically finds all the dependencies for a given state, and these need not be of a fixed order, since the mined patterns can be arbitrarily long. Moreover, the elements of these patterns do not need to be consecutive, i.e., a variable length gap could exist between the elements. This enables us to model multiple higher order HMMs via a single variable-order state machine that executes faster and yields much greater accuracy. This contribution is composed of (i) a **Variable Gap Sequence (VGS)** miner, which is a contribution in the area of pattern extraction. VGS mines frequent patterns with different lengths and gaps between the elements across and within several sequences. VGS can be used individually as well as part of VOGUE for pattern extraction, and (ii) a VOGUE state machine that uses the mined variable-gap sequences from VGS to model multiple higher order HMMs via a single variable-order state machine.

Moreover, we applied VOGUE to a real world problem, namely, finding homologous proteins. Although VOGUE has a much wider applicability, such as in web accesses, text mining, user behavior analysis, etc, in this work we apply VOGUE to a real world problem in biological sequence analysis, namely, multi-class protein classification. Given a database of protein sequences, the goal is to build a statistical model so that we can determine whether a query protein belongs to a given family (class) or not. Statistical models for proteins, such as profiles, position-specific scoring matrices, and hidden Markov models [29] have been developed to find homologs. However, in most biological sequences, interesting patterns repeat (either within the same sequence or across sequences) and may be separated by variable length gaps. Therefore a method like VOGUE that specifically takes these kind of patterns into consideration can be very effective. We show experimentally that VOGUE's modeling power is superior to higher-order HMMs while reducing the latter's state-space complexity, and improving their prediction. VOGUE also outperforms HMMER [29], a HMM model especially designed for protein sequences.

The second contribution is in the area of data interpretation and decoding. This contribution is a consequence of the unique structure of VOGUE sate machine, where the gaps have a notion of duration. Therefore, we adjusted the widely used Viterbi algorithm, that solves the interpretation problem, to meet those needs. We call this method Variable-Gap Viterbi (VG-Viterbi). We optimized VG-Viterbi based on the fact that the transition matrix between the states of the model is a sparse matrix and so there is no need to model the transitions between all the states.

The third contribution is Canonical VOGUE (C-VOGUE) that aims at increasing the already "good" performance of VOGUE by eliminating artifacts in the extracted patterns, hence reducing the number of patterns to be modeled later on. These artifacts are retained as being frequent patterns but each one of these patterns is in fact an artifact of another pattern. This contribution aims at decreasing the state space complexity of the state machine, which is a major step towards one of the three goals of modeling with state machines while keeping good accuracy and coverage.

VOGUE is adaptable enough to allow for inclusion of domain specific knowledge to better model patterns with higher order structures unlike other techniques that are made specially for 1 dimensional patterns, and perform poorly. We achieved this by S-VOGUE (Substitution VOGUE), where the mined patterns are chosen not only based on the frequency of exact match items but also among items that could be substituted by one another according to their secondary or tertiary structure. This is, in fact, very helpful in protein analysis where proteins of the same family share common patterns (motifs) that are not based on exact match but rather on substitutions based on the protein sequences elements weight, charge, and hydrophobicity. These elements are called amino acids.

II. Pattern Mining Using Sequence Mining

Data mining involves the process of analyzing data to show patterns or relationships; sorting through large amounts of data; and picking out pieces of relative information or patterns that occur e.g., picking out statistical information from some data [30]. There are several data mining techniques, such as association rules, sequence mining [74], classification and regression, similarity search and deviation detection [34], [41], [42], [81]–[83]. Most of the real world applications encompass sequential and temporal data. For example, analysis of biological sequences sush as DNA, proteins, etc. Another example is in web prefetching, where pages are accessed in a session by a user in a sequential manner. In this type of data each "example" is represented as a sequence of "events", where each event might be described by a set of attributes. Sequence Mining helps to discover frequent sequential attributes or patterns across time or positions in a given data set. In the domain of web usage, a database would be the web page accesses. Here the attribute is a web page and the object is the web user. The sequences of most frequently accessed pages are the discovered "frequent" patterns. Biological sequence analysis [36], [92], identifying plan failures [90], and finding network alarms [43], constitute some of the real world applications where sequence mining is applied.

In this work, we choose to use sequence mining rather than association mining due to the fact that association mining discovers only intra-itemsets patterns where items are unordered, while sequence mining discovers inter-itemsets, called sequences, where items are ordered [89].

TABLE I

Original Input-Sequence Database

SID	$Time(EID)$	$items$
1	10	AB
1	20	A
1	30	AB
2	20	AC
2	30	ABC
2	50	B
3	10	A
3	30	B
3	40	A
4	30	AB
4	40	A
4	50	B

TABLE II

Frequent 1-Sequences ($min\_sup = 3$)

A	4
B	4

TABLE III

FREQUENT 2-SEQUENCES ($min\_sup = 3$)

AB	3
$A \rightarrow A$	4
$A \rightarrow B$	4
$B \rightarrow A$	3
$B \rightarrow B$	3

TABLE IV

FREQUENT 3-SEQUENCES ($min\_sup = 3$)

AB	3
$AB \rightarrow B$	3

A. Sequence Mining discovery: Definitions

The problem of mining sequential patterns, as defined in [4] and [74], is as follows: Let's consider $I = \{I_1, \cdots I_m\}$ be the set of m distinct items. An *itemset* is a subset of I with possibly un-ordered items. A *sequence S*, on the other hand, is an ordered list of itemsets from I (i.e., $S = I_1, \cdots I_l$ where $I_j \subseteq I$ and $1 \leq j \leq l$). S can be defined as $S = I_1 \rightarrow I_2 \cdots \rightarrow I_l$, where "$\rightarrow$" is a "happen after" relationship denoted as $I_i \preceq I_j$ and $i \leq j$. The length of the sequence S is defined as $\mid S \mid = \sum_j \mid I_j \mid$, where $\mid I_j \mid$ is the number of items in the itemset I_j. For example, let's consider the sequence $S = AB \rightarrow C \rightarrow DF$. This sequence is composed of 3 itemsets, namely, AB, C, and DF, and its length is $\mid S \mid = \mid AB \mid + \mid C \mid + \mid DF \mid = 5$. The sequence S is then called *5-sequence*. We will refer for the remaining of this proposal to a sequence of length *k* as *k-sequence*.

A sequence $S' = I'_1, \cdots I'_p$ is called a subsequence of *S*, with $p \leq \mid S \mid$, if there exist a list of itemsets of *S*, $I_{i_1}, \cdots I_{i_p}$ such that $I_j \subseteq I'_{i_k}$, $1 \leq k, j \leq p$. For example, the sequence $S' = A \rightarrow D$ is a subsequence of *S* (described in the previous example), because $A \subseteq AB$, $D \subseteq DF$, and the order of itemsets is preserved. Let *D*, a set of sequences, be a sequential database, where each sequence $S \in D$ has a unique identifier, denoted as *sid*, and each itemset in *S* has a unique identifier, denoted as *eid*. The *support* of *S'*, is defined as the fraction of the database sequences that contain *S'*, given as $\sigma_D(s') = \mid D_s \mid / \mid D \mid$, where $D_{s'}$ is a set, contained in *D*, of database sequences *S* such that $S' \subseteq S$. A sequence is said to be frequent if it occurs more than a user-specified threshold $minsup$, called minimum support. The problem of mining frequent patterns is to find all frequent sequences in the database. This is formally defined in [74] as:

Definitions for Sequential Pattern Mining: *Given a sequential database D and a user-specified* $minsup$*parameter* σ *(*$0 \leq \sigma \leq 1$*), find all sequences each of which is supported by at least* $\lceil \sigma \mid D \mid \rceil$ *of sequences in* D. F_k denotes the set of frequent k-sequences and **Maximal frequent sequence** is a sequence that is not a subsequence of any other frequent sequence.

Table I shows an example database [88]. It consists of three items (A, B, C), four input sequences and twelve events. Tables II, III and IV show the frequent 1-sequences, 2-sequences, and 3-sequences with a *min_sup* of 3, corresponding to 75% of the data, respectively. The *maximal* sequences are $A \rightarrow A$, $B \rightarrow A$, and $AB \rightarrow B$.

B. Sequence Mining discovery: Related Work

It is challenging to find all frequent patterns in a large database where the search space becomes extremely large. In fact, there are $O(m^k)$ possible frequent sequences of length at most k, where m is the number of different symbols in the database alphabet.

Many techniques have been proposed to mine temporal data sets to extract the frequent sequences. However, if the search is unconstrained, it can produce millions of rules. Moreover, some constrains might need to be added in some domains. For example, a user might be interested in searching for sequences occurring close in time to each other or far apart from each other, those that contain some specific items, occurring during a specific period of time, or frequent at most a number of times or at least another number of times in the data set.

Several techniques have been proposed to discover the frequent sequences [5], [55], [62], [90]. One of the early algorithms that efficiently discovered the frequent sequences is the AprioriAll [74], that iteratively finds itemsets of length l based on previously generated *(l-1)*-length frequent itemsets. In [45] frequent sequences in a single long input-sequence, called frequent episodes, were mined. It was extended to discover generalized episodes that allow uniary conditions on individual sequences itemsets, or binary conditions on itemset pairs [54]. In [3], the *Generalized Sequential Patterns* (*GSP*) algorithm was proposed to extend the *AprioriAll* algorithm by introducing user-specified minimum gap and maximum gap time constraints, user-specified sliding window size, and user-specified minimum support. GSP is an iterative algorithm that counts candidate frequent sequences of length k in the $k - th$ database scan. However, *GSP* suffers from a number of drawbacks, namely, it needs as many full scans of the database as the longest frequent sequence; it uses a complex hash structure with poor locality; and it scales up linearly as the size of the data increases. *SPADE* [89] was proposed to cope with *GSP*'s drawbacks. SPADE uses a vertical id-list database, prefix-based equivalence classes, and it enumerates frequent sequences through simple temporal joins. *SPADE* uses dynamic programming concepts to break the large search space of frequent patterns into small and independent chunks. It requires only three scans over the database as opposed to *GSP* which requires multiple scans, and *SPADE* has the capacity of in-memory computation and parallelization which can considerably decrease the computation time. *SPADE* was later on extended to Constraint SPADE (*cSPADE*) [88] which considers constraints like max/min gaps and sliding windows. SPIRIT [39] is a family of four algorithms for mining sequences that are a complementary to cSPADE. However, cSPADE considers a different constraint that finds sequences predictive of at least one class for temporal classification problems. In fact, SPIRIT mines sequences that match user-specified regular-expression constraints. The most relaxed of the four is SPIRIT(N) that eliminates items not appearing in any of the user specified regular-expressions. The most strict one is SPIRIT(R), that applies the constraints, while mining and only outputs the exact set. GenPresixSpan [6] is another algorithm based on PrefixSpan [64] that considers gap-constraints. Regular expressions and other constraints have been studied in [40], [48], [92]. In [48], a mine-and-examine paradigm for interactive exploration of association and sequence episodes was presented, where a large collection of frequent patterns is first to be mined and produced. Then the user can explore this collection by using "templates" that specify what is of interest and what is not. In [60], CAP algorithm was proposed to extract all frequent associations matching a large number of constraints. However, because of the constrained associations, these methods are unsuitable for temporal sequences that introduce different kinds of constraints.

Since *cSPADE* is an efficient and scalable method for mining frequent sequences, we will use it as a base for our new method Variable-Gap Sequence miner (VGS), to extract the patterns that will be used to estimate the parameters and structure of our proposed VOGUE state machine. The main difference, however, between VGS and cSPADE is that cSPADE essentially ignores the length and symbol distributions of gaps, whereas VGS is specially designed to extract such patterns within one or more sequences. Note that while other methods can also mine gapped sequences [6], [40], [92], the key difference is that *during mining* VGS explicitly keeps track of all the intermediate symbols, their frequency, and the gap frequency distribution, which are used to build VOGUE.

Before we go into more details about cSAPDE, we will give an overview of SPADE algorithm in Section 1, since cSPADE is an extension of it. *cSPADE* technique will be described in the Section 2. Section III-B provides the details of our proposed method VGS.

III. SPADE: Sequential Patterns Discovery using Equivalence classes

The SPADE algorithm [89] is developed for fast mining of frequent sequences. Given as input a sequential database *D* and the minimum support, denoted as *min_sup*, the main steps of SPADE consists of the following:

1) Computation of the frequent *1-sequences*: $F_1 = \{$ frequent items or *1-sequences*$\}$;
2) Computation of the frequent *2-sequences*: $F_2 = \{$ frequent items or *2-sequences* $\}$;
3) Decomposition into prefix-based parent equivalence classes: $\xi = \{$ equivalence classes $[X]_{\theta_1}\}$;
4) Enumeration of all other sequences, using *Breadth-First Search* (*BFS*) or *Depth-First Search* (*DFS*) techniques, within each class $[X]$ in ξ.

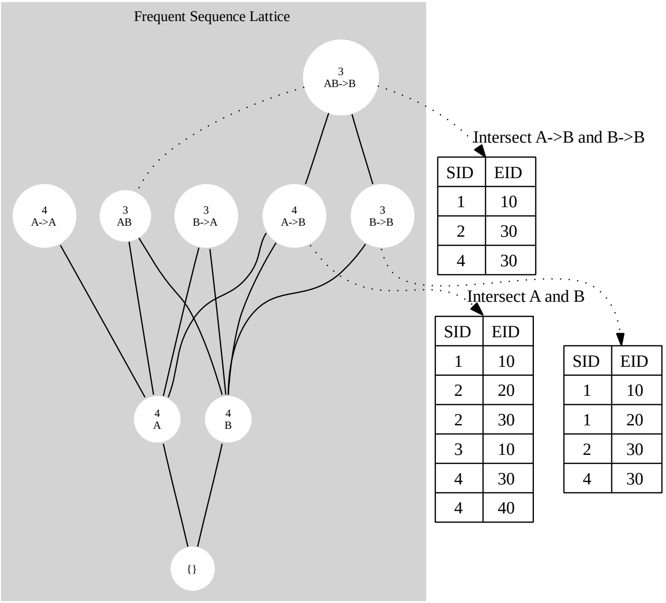
In the above, *i-sequences* denotes sequences of length i, $1 \leq i \leq 2$.

A formal description of SPADE [89] is given in Algorithm 1. The SPADE algorithm uses the following concepts:

Algorithm 1 SPADE

```
procedure SPADE($min\_sup$)
    $P = \{$ parent classes $P_i\}$;
    for each parent class $P_i \in P$ do
        Enumerate-Frequent-Seq($P_i$);
    end for
end procedure
function ENUMERATE-FREQUENT-SEQ(S)
    for all atoms $A_i \in S$ do
        $T_i = \emptyset$;
        for all atoms $A_j \in S$ with $j \geq i$ do
            $R = A_i \vee A_j$;
            if $Prune(R) == FALSE$ then
                $L(R) = L(A_i \cap L(A_j))$;
                if $\sigma(R) \geq min_s up$ then
                    $T_i = T_i \cup \{R\};] F_{|R|} = F_{|R|} \cup \{R\}$;
                end if
            end if
        end for
        if (Breadth-First-Search) then
            Enumerate-Frequent-Seq($T_i$);
        end if
    end for
    if (Breadth-First-Search) then
        for all $T_i \neq \emptyset$ do
            Enumerate-Frequent-Seq($T_i$);
        end for
    end if
end function
```

Sequence Lattice: If a sequence S is frequent all subsequences S' of S, such that $S' \preceq S$, are frequent. In fact SPADE considers that the subsequence relation $\preceq$ is a partial order on the set of sequences. Therefore, SPADE finds the subsequences that are frequent from the most *general*, single items, to the most specific, the maximal sequences in either a depth-first-search or breath-first-search manner. This is done by looking into the sequence lattice spanned by the subsequence ($\preceq$) relation as shown in Figure 1 for the example dataset.



SID	EID
1	10
2	30
4	30

SID	EID
1	10
2	20
2	30
3	10
4	30
4	40

SID	EID
1	10
1	20
2	30
4	30

Fig. 1. Frequent Sequence Lattice and Temporal Joins.

Support Counting: One of the main differences between SPADE and the other sequence mining algorithms [5], [74] is that the latter ones consider a *horizontal database* layout whereas SPADE considers a vertical one. The database in the horizontal format consists of a set of input sequences which in their turn consist of a set of events and the items contained in the events. The vertical database, on the other hand, consists of an a disk-based id-list, denoted $L(X)$ for each item X in the sequence lattice, where each entry of the id-list is a pair of sequence id and event id (sid, eid) where the item occurs. For example, let's consider the database described in Table I, the id-list consist of the tuples $\{(2, 20), (2, 30)\}$.

With the vertical layout in mind, the computation of F_1 and F_2 becomes as follows:

Computation of F_1: The id-list of each database item is read from disk into memory. Then the id-list is scanned incrementing the new sid encountered. All this is done in a single database scan.

Computation of F_2: Let $N = |F_1|$ be the number of frequent items, and *A* the average id-list size in bytes. In order to compute F_2 a naive implementation will require $\binom{N}{2}$ id-list joins for all pairs of items. Then, $\left(\frac{A \times N \times (N-1)}{2}\right)$ is the corresponding amount of data read; this is almost $N/2$ data scans. To avoid this inefficient method two alternate solutions were proposed in [88]:

To compute F_2 above a user specified lower bound threshold, a preprocessing step is used. An on-the-fly vertical-to-horizontal transformation is performed: scan the id-list of each item i into memory. Then for each (sid, eid) pair (i, e) is inserted in the list for input sequence S. Using the id-list for item A from the previous example in Table I, the first pair $(1, 15)$ is scanned then $(A, 15)$ is inserted in the list for input-sequence 1. Table V describes the complete vertical-to-horizontal transformation of the database. To recover the horizontal database, for each *sid*, a list of all $2-sequences$ is formed in the list, and counts are updated in a $2-dimensional$ array indexed by the frequent items.

Then, as shown in Figure 1, the intermediate id-list for $A \rightarrow B$ is obtained by a temporal join on the lists of A and B. All occurrences of A "before" B, that represent $A \rightarrow B$ are found

TABLE V
VERTICAL-TO-HORIZONTAL DATABASE RECOVERY

sid	$(item, eid) pairs$
1	$(A,10)(A,30)(B,10)(B,20)(B,30)$
2	$(A,20)(A,30)(B,30)(B,50)(C,20)(C,30)$
3	$(A,10)(A,40)(B,30)$
4	$(A,30)(A,40)(B,30)(B,50)$

in an input sequence and the corresponding eids are stored to obtain $L(A \rightarrow B)$. In the case of $AB \rightarrow B$, the id-lists of its two generating sequences $A \rightarrow B$ and $B \rightarrow B$ are joined. Because of main-memory limitations, it is not possible to enumerate all the frequent sequences by traversing the lattice and performing joins. However, this large search space is decomposed by SPADE into smaller chunks, called *classes*, to be processed separately by using *suffix-based equivalence classes*.

Definition: Two k-sequences are in the same class if they share a common $k-1$ length suffix.

Therefore, each class is a *sub-lattice* of the original lattice. It can be processed independently since it contains all the information to generate all frequent sequences with the same suffix. SPADE recursively calls the procedure *Enumerate-Frequent* that counts the suffix classes starting from suffix-classes of length one (called *parent classes*), in the running example (A, B), then it uses suffix-classes of length two, in the running example ($A \rightarrow B$, AB) and so on. The input to the procedure is a set of items of a sub-lattice S, along with their id-lists. The id-lists of all distinct pairs of sequences in each class are joined to generate the frequent sequences and the results is checked against the user set threshold min_sup.

Temporal Joins: A *suffix equivalence class* $[S]$ can contain either an itemset of the form XS or a sequence of the form $Y \rightarrow S$, where X and Y are items, and S is a *suffix* sequence. Assuming that itemsets of a class always precede its sequences, then joining the id-lists of all pairs of elements extends the class for the next level. This results in one of the three different frequent sequences depending on the joined pairs [88]:

1) *Joining an Itemset to another Itemset*: the resulting sequence is an itemset. For example, AS with BS the resulting sequence is the itemset ABS.
2) *Joining an Itemset to a Sequence*: the resulting sequence is a new sequence. For example, AS with $B \rightarrow S$ results in the sequence $B \rightarrow AS$.
3) *Joining a Sequence to another Sequence*: there are three possible resulting sequences considering the sequences $A \rightarrow S$ and $B \rightarrow S$: a new *itemset* $AB \rightarrow S$, the sequence $A \rightarrow B \rightarrow S$ or the sequence $B \rightarrow A \rightarrow S$.

From Figure 1, from the $1-sequences$ A and B we can get three sequences: the itemset AB and the two sequences $A \rightarrow B$ and its "reverse" $B \rightarrow A$. To obtain the id-list of itemset AB, the equality of (sid,eid) pairs needs to be checked and in this case it is $L(AB) = \{(1,10),(1,30),(2,20),(4,30)\}$ in Figure 1. This shows that AB is frequent in 3 out of the 4 sequences in the data set ($min\_sup = 3$ which corresponds to 75% of the data). In the case of the resulting sequence $A \rightarrow B$, there is need to check for (sid,eid) pairs where sid for both A and B are the same but the eid for B is strictly greater in time than the one for A. The (sid,eid) pairs in the resulting id-list for $A \rightarrow B$ only keep the information about the first item A and not B. This is because all members of a class share the same suffix and hence the same eid for that suffix.

A. cSPADE: ***c**onstrained **S**equential **P**atterns **D**iscovery using **E**quivalence classes*

In this section we describe in some detail the cSPADE algorithm [88]. cSPADE is designed to discover the following types of patterns: (i) Single item sequences as well as the sequences of subsets of items. For example the set AB, and $AB \rightarrow C$ in $(AB \rightarrow C \rightarrow DF)$. (ii) Sequences with variable gaps among itemsets (a gap of 0 will discover the sequences with consecutive itemsets). For example, from the sequence $(AB \rightarrow C \rightarrow DF)$, $AB \rightarrow DF$ is a subsequence of gap 1 and $AB \rightarrow C$ is a subsequence of gap 0.

cSPADE is an extension of the **S**equential **P**atterns **D**iscovery using **E**quivalence classes (SPADE) algorithm by adding the following constraints:

1) Length and width restrictions.
2) Minimum gap between sequence elements.
3) Maximum gap between sequence elements.
4) A time window of occurrence of the whole sequence.
5) Item constraints for including or excluding certain items.
6) Finding sequences distinctive of at least a special attribute-value pair.

Definition: A Constraint [88] is said to be *class-preserving* if in the presence of the constraint *suffix-class* retains it's self-containment property, $i.e.$, support of any k-sequence can be found by joining id-lists of its two generating subsequences of length $(k-1)$ within the same class.

If a constraint is *class-preserving* [88], the frequent sequences satisfying that constraint can be listed using local suffix class information only. Among all the constraints stated above, the *maximum gap* constraint is the only one that is not class-preserving. Therefore, there is a need for a different enumeration method. cSPADE pseudo-code is described in Algorithm 2.

Algorithm 2 cSPADE

```
procedure cSPADE($min\_sup$)
    $P = \{$ parent classes $P_i\}$;
    for each parent class $P_i \in P$ do
        Enumerate-Frequent-Seq($P_i$);
    end for
end procedure
function ENUMERATE-FREQUENT-SEQ(S)
    for all sequences $A_i \in S$ do
        if maxgap join with $F_2$ then
            $p =$ Prefix-Item($A_i$);
            $N = \{$ all sequences $A_j$ in class $[p]\}$;
        else if  self-join then
            $N = \{$ all sequences $A_j \in S$, with $j \geq i\}$;
        end if
        for all sequences $\alpha \in N$ do
            if $(length(R) \leq max_l)$ and $width(R) \leq max_w)$ and $accuracy(R) \neq 100\%)$ then
                $L(R)$ = Constrained-Temporal-Join($L(A_i), L(\alpha)$, min-gap, max-gap, window);
                if $\sigma(R, c_i) \geq min_sup(c_i)$ then
                    $T = T \cup \{R\}$; print $R$;
                end if
            end if
        end for
    end for
    Enumerate-Frequent($T$);
    delete S;
end function
```

We will describe in some more detail how cSPADE handles each one of those constraints:

Length and Width restrictions: Without a maximum length allowed for a pattern to be mined, the number of frequent sequences blows up especially in the case some items are very frequent in highly structured data sets. In cSPADE this is taken care of by adding the following check [88]: if *width*$(R) \leq max_w$ and if *length*$(R) \leq max_l$, where max_w and max_l are, respectively the maximum width and length allowed in a sequence. This addition is done in the "Enumerate" method as shown in cSPADE's pseudo-code. These constraints are *class-preserving* since they do not affect id-lists.

Minimum Gap between sequence elements: Patterns, which items are not necessarily immediately consecutive in a sequence, are very important in some domains such as in DNA analysis. The minimum gap is a *class-preserving* constraint. In fact, if we consider that a sequence $A \rightarrow B \rightarrow S$ is frequent with a min-gap of at least δ between each two of its elements, then A and S are at least δ elements apart and the same goes for B and S. Therefore, by joining the id-lists of the two sequences $A \rightarrow S$ and $B \rightarrow S$ one can determine if $A \rightarrow B \rightarrow S$ is frequent. Hence, adding the constraint minimum gap boils down to adding a check in SPADE pseudo-code for the minimum gap between the items of a sequence. If we consider the example data set in Figure 1, the lattice $L(A \rightarrow B)$ is generated by adding a check on the (sid,eid) pairs in $L(A)$ and $L(B)$. In fact, for every given pair (c, t_b) in $L(A)$ we check if there exist a pair (c, t_b) in $L(B)$ that satisfies the constraint $t_b \neq t_a$ and $t_b - t_a \geq min\_gap$. If that is the case the pair (c, t_a) is added to $L(A \rightarrow B)$. For example, if min_gap is set to 20 then the pair $(1, 10)$ from $L(A)$ can be added to $L(A \rightarrow B)$ since there exist a pair $(1, 30)$ that satisfies the constraint [88].

Maximum Gap between sequence elements: This constraint is not *class-preserving* since if there is a sequence $A \rightarrow B \rightarrow S$ is frequent with $max\_gap = \delta$, then the subsequence $B \rightarrow S$ is frequent with at most $max\_gap = \delta$ between B and S but $A \rightarrow S$ is only frequent at most $max\_gap = 2\delta$. Therefore, if $A \rightarrow S$ could be infrequent with this constraint but yet $A \rightarrow B \rightarrow S$ is frequent. To incorporate maximum gap constraint to the SPADE pseudo-code [88], first a check needs to be added such that, in the example of Figure 1, for a given pair (c, t_a) in $L(A)$, check if a pairs (c, t_b) exists in $L(B)$ such that $t_b \neq t_a$ and $t_b - t_a \leq max\_gap$. The second step is to change the enumeration of the sequences with the maximum gap constraint. A join with F_2 is necessary instead of a self-join because the classes are no more self-contained. This join is done recursively with F_2 until no extension is found to be frequent.

Time Window of occurrence of the whole sequence: In other words, the time constraint applies to the whole sequence instead of minimum or maximum gap between elements of the sequence [88]. This constraint is *class-preserving* since if a sequence $A \rightarrow B \rightarrow S$ is within a time-window δ, then it implies that $A \rightarrow S$ and $B \rightarrow S$ are within the same window and so on for any subsequence. However, including the time-window into the SPADE software is difficult because the information concerning the whole sequence time is lost from the parent class. In fact, only the information about the eid of the first item of the sequence is stored and the one of the remaining items is lost from one class to the next. The solution proposed in [88] is to keep information about the difference "diff" between the eid of the first and the last item of the sequence at each step of the process. This is done by adding an extra column in the id-list called *diff* to store that information.

Item constraints for including or excluding certain items: The use of a vertical format of the data set and the equivalence classes in cSPADE makes it easy to add constraints on items within sequences [88]. For instance, if the constraint is excluding a certain item from the frequent sequences, then removing it from parent classes takes care of that. Therefore, no frequent sequence will contain that item. The same procedure will apply in the case of including an item.

Whereas cSPADE essentially ignores the length and symbol distributions of gaps, the new mining sequence algorithm that we present in the work, *VGS* (Variable-Gap Sequences), is specially designed to extract such patterns within one or more sequences.

TABLE VI

VGS: SUBSEQUENCES OF LENGTH 1

	A	B	C	D	E	F	G	H	I
frequency	4	3	2	2	1	1	1	1	1

TABLE VII

VGS: SUBSEQUENCES OF LENGTH 2

subsequence	*freq*	$g = 0$	$g = 1$	$g = 2$
$A \rightarrow C$	2	1	1	0
$A \rightarrow B$	3	0	1	2
$A \rightarrow D$	2	1	0	1
$C \rightarrow B$	2	2	0	0
$C \rightarrow D$	2	0	1	1
$C \rightarrow A$	2	0	1	1
$B \rightarrow D$	2	1	1	0
$B \rightarrow A$	2	1	1	0
$D \rightarrow A$	2	1	0	1

The next Chapter describes *VGS* in more details.

B. Pattern Extraction: ***V****ariable-****G****ap* ***S****equence (****VGS****) miner*

Variable-Gap Sequence miner (VGS) is based on cSPADE [88]. While cSPADE essentially ignores the length and symbol distributions of gaps, VGS is specially designed to extract such patterns within one or more sequences. Note that whereas other methods can also mine gapped sequences [6], [74], [88], [92], the key difference is that *during mining* VGS explicitly keeps track of all the intermediate symbols, their frequency, and the gap frequency distributions, which are used to build VOGUE.

VGS takes as input the maximum gap allowed ($maxgap$), the maximum sequence length (k), and the minimum frequency threshold (min_sup). VGS mines all sequences having up to k elements, with no more than $maxgap$ gaps between any two elements, such that the sequence occurs at least min_sup times in the data. For example, let $S = ACBDAHCBADFGAIEB$ be an input sequence over the alphabet $\Sigma = \{A, \cdots, I\}$, and let $maxgap = 2$, $minsup = 2$ and $k = 2$. VGS first mines the frequent subsequences of length 1, as shown in Table VI. Those symbols that are frequent are extended to consider sequences of length 2, as shown in Table VII. For example, $A \rightarrow B$ is a frequent sequence with frequency $freq = 3$, since it occurs once with gap of length 1 (ACB) and twice with a gap of length 2 ($AHCB$ and $AIEB$). Thus the gap length distribution of $A \rightarrow B$ is $0, 1, 2$ as shown, in Table VII, under columns $g = 0$, $g = 1$, and $g = 2$, respectively. VGS also records the symbol distribution in the gaps for each frequent sequence. For $A \rightarrow B$, VGS will record gap symbol frequencies as $C(2), E(1), H(1), I(1)$, based on the three occurrences. Since $k = 2$, VGS would stop after mining sequences of length 2. Otherwise, VGS would continue mining sequences of length $k \geq 3$, until all sequences with k elements are mined.

Before we start describing VGS, we will provide definitions of some terms that will be used in this section and in the rest of this thesis:

k**-seq**: sequence of length k, i.e. k elements. For example, $A \rightarrow B$ is 2-seq where B occurs after A and $A \rightarrow B \rightarrow C$ is a 3-seq and so on.

min_sup: minimum support, is the minimum threshold for the frequency count of sequences.

maxgap: maximum gap allowed between any two elements of a k-seq.

F_1: the set of frequent 1-seq (single items).

F_k: the set of all k-seq which frequency is higher than the minimum threshold $\mathbf{min\_sup}$ and the gap between their elements is at most of length ***maxgap***.

C_k: the set of candidate k-seq.

TABLE VIII

Subsequences of length 2 mined by VGS

subsequence	*freq*	$g = 0$	$g = 1$	$g = 2$
$A \rightarrow C$	2	1	1	0
$A \rightarrow B$	3	0	1	2
$A \rightarrow D$	2	1	0	1
$C \rightarrow B$	2	2	0	0
$C \rightarrow D$	2	0	1	1
$C \rightarrow A$	2	0	1	1
$B \rightarrow D$	2	1	1	0
$B \rightarrow A$	2	1	1	0
$D \rightarrow A$	2	1	0	1

TABLE IX

Id-list for the item A

SID	EID
1	1
1	5
1	9
1	13

The first step of VOGUE uses VGS to mine the raw data-set for variable gap frequent sequences. It takes as input the maximum gap allowed maxgap between the elements of the subsequences, the maximum length (k) of the subsequences, and the minimum frequency threshold (min_sup) for sequences to be considered frequent. The mined subsequences from VGS are of the form $A \rightarrow B$ if k $= 2$, or $A \rightarrow B \rightarrow C$ if k $= 3$, and so on. The frequency of the subsequences is calculated either across the sequences in the data-set and/or within the sequences in the data-set, as the application may require. Only those sequences that are frequent within a gap range of $[0, maxgap]$ are reported. As a running example, let $S =< A, C, B, D, A, H, C, B, A, D, F, G, A, I, E, B >$ be a sequence. Let $maxgap = 2$, $min\_sup = 2$ and $k = 2$. The results of VGS are shown in Table VII. For example, $A \rightarrow B$ is a frequent sequence with $freq = 3$, since we have $< A, C, B >$ for a gap of 1, and $< A, H, C, B >$ and $< A, I, E, B >$ for a gap of 2. In the table, the columns $g = 0$, $g = 1$, and $g = 2$ show the gap distributions. For this subsequence we have no occurrences at $g = 0$, one at $g = 1$ and two at $g = 2$.

The key features of our approach are: (i) Use of a vertical id-list database format, where each sequence is associated with a list of objects in which it occurs, and its relative time-stamps, and (ii) employeed a lattice-theoretic approach to decompose the original search space (lattice) into smaller sub-lattices which can be processed independently in the main memory. This reduces the I/O cost, since the algorithm requires only three scans of the data set. Refer to [21] for a detailed introduction to Lattice theory.

VGS is, therefore, cost efficient by reducing the dataset scans and using an efficient depth first search, as described in [89]. We use, as in [89], a vertical database format, where an id-list for each item in the dataset.Each entry in the id-list is a(sid, eid) pair. Eid is where the item exists in sequence which id is sid. sid is the sequence id in the data set and eid is the event id where the item exists.

In our example we have 9 different items $\{A, B, C, D, E, F, G, H, I\}$. The corresponding id-list of A is shown in Table IX. This allows checking the frequency of the sequences via joins of the items. Using our running example, the join of A and B would be $A \vee B = \{A \rightarrow B, B \rightarrow A\}$; this give us the maximal sub-sequences existing in the data set formed by A and B with a maximum gap length g of $maxgap$.

The main steps in VGS are:

Computation of F_1: we compute all frequent 1-seq (single items) in the whole data set and their frequencies regardless of the $minsup$. This is done by reading the id-list of each item and scanning it incrementing the support for each sid encountered, even if it repeats, for each sid. This is different from SPADE where only new sid are taken into considerations to look for patterns across only the sequences. In VGS we look for patterns within and across the sequences in the data set.

Computation of F_2: we compute all the 2-seq with a gap of length g between its elements, $g \in \{0, \cdots, maxgap\}$ in which frequencies are greater than the min_sup. $g = 0$ corresponds to no elements between two main elements of the k-seq, and $g = 1$ corresponds to one element between two main elements of the elements of the k-seq and so on. This computation is done by scanning the id-list of each item in the alphabet into memory. For each pair (sid, eid) we insert it in the list for the input sequence whose id is sid. We, then, form a list of all 2-sequences in the list for each sid, and increment the frequency if the difference between the two eid events is less than the $maxgap$ allowed.

Enumeration: of all other frequent k-seq, with frequency at least min-sup, with variable gaps between each two of its elements via Depth First Search (DFS) within each class. For example, the 3-sequence $A \xrightarrow{g_1} B \xrightarrow{g_1} C$ has with gap $g_1 \in \{0, \cdots, maxgap\}$ between A and B and gap $g_2 \in \{0, \cdots, maxgap\}$ between B and C. Where “$A \overset{g_1}{\to} B$” means A is followed by B after g_1 elements in between them. Likewise, “$B \overset{g_2}{\to} C$” means B is followed by C after g_2 elements in between them. The procedure is described in Algorithm 3.

Algorithm 3 VGS

```
procedure VGS($min\_sup$,maxgap)
    $P = \{$ parent classes $P_i\}$;
    for each parent class $P_i \in P$ do
        Enumerate-Frequent-Seq($P_i$,$min\_sup$,maxgap);
    end for
end procedure
function ENUMERATE-FREQUENT-SEQ(S, $min\_sup$,maxgap)
    for all items $v_i \in S$ do
        $T_i = \emptyset$;
        for all items $v_j \in S$ with $j \geq i$ do
            $R =$ new candidate sequence from $v_i$ and $v_j$;
            $L(R) = L(v_i) \cap L(v_j)$;        ▷ with
    $0 \leq (v_i(eid) - v_j(eid)) \leq maxgap$, where $v_i(eid)$ is the event
    id of $v_i$
            if $freq(R) \geq minsup$ then
                $T_i = T_i \cup \{R\}$;
                $F_{|R|} = F_{|R|} \cup \{R\}$;
            end if
        end for
        Enumerate-Frequent-Seq($T_i$);
    end for
end function
```

The input to the procedure is a set of items of a sub-lattice S, along with their id-lists and the min_sup and $maxgap$. The sequences that are found to be frequent form the atoms of classes for the next level. This process is done recursively until all the frequent sequences are computed.

IV. Data Modeling Using Variable-Order State Machine

HMMs are a powerful statistical tool that have been applied in a variety of problems ranging from speech recognition, to biological sequence analysis, to robot planning, to web prefetching. Speech recognition, however, is the area of research where a considerable amount of papers and books on using HMM have been produced [51], [70]. The best description of how HMMs have been used in Speech recognition is described in the well referenced tutorial by Rabiner [67]. As biological knowledge accumulates, HMMs have been used as one of the statistical structures for biological sequence analysis, a growing field of research, from human genomes to protein folding problems, [26]. In [28], Profile HMMs have been used for multiple alignment of conserved sequences. HMMs have been used as well in inferring phylogenetic trees [63], and in splice site detection [44]. Partially Observable Markov Decision Process (POMDP) models have been used in robot planning to allow the robots to act and learn even if they are uncertain about their current location, [49]. In [66], all K^{th} Markov model have been used to predict web surfing behavior. A hidden markov model was defined for each character in a text recognition application from grey level images in [1]. In [56], an HMM was used for automatic gait classification in medical image processing. We assume the reader is familar with HMM terms and definitions.

There are three basic problems to solve while applying HMMs to real world problems:**(i) Evaluation:** Given the observation sequence $O = o_1 o_2 \cdots o_T$, and a model $\lambda = (A, B, \Pi)$, how do we efficiently compute $P(O|\lambda)$? The solution to this problem will enable us to evaluate different models and choose the best one according to the given observation. The main issue, however, in this problem is that the hidden states tend to complicate the evaluation. This problem can be solved using the Forward-Backward algorithm [67]. **(ii) Decoding:** Given the observation sequence $O = \{o_1 o_2 ... o_T\}$, and the model λ, how do we choose a corresponding state sequence $Q = \{q_1 q_2 ... q_T\}$ which is optimal in some meaningful sense? The solution to this problem would explain the data. An optimization criterion has to be decided (e.g., maximum likelihood). An efficient dynamic programming method, as the Viterbi algorithm [67], is used to solve this problem. **(iii) Learning:** How do we adjust the model parameters $\lambda = (A, B, \Pi)$ to maximize $P(O|\lambda)$? This problem is about finding the best model that describes the observation at hand. Of all the three problems, the third one is the most crucial and challenging to solve for most applications of HMMs. The work presented here focuses on this problem.

The main problem with HMMs is that they depend on the Markov property, i.e., the current state depends only on the previous state. In general patterns may display longer range dependencies, and one needs a higher-order HMM [25] (where the order denotes the number of previous states the current state depends on) to model such patterns. Thus, in addition to HMMs (of order 1), there are other types of Markov models used in prediction. For example, an $m-$order Prediction-by-Partial Match (PPM) predictor maintains a Markov model of order j, for all $1 \leq j \leq m$. This scheme is also referred to as an *All-m-Order Markov Model* [58]. This model uses the past j events to compute the probability of next event to come. Mixed-order HMMs have also been proposed [72]. The main challenge here is that building higher order HMMs [25] is not easy, since we have to estimate the joint probabilities of the previous m states (in an m-order HMM). Furthermore, not all of the previous m states may predict the current state. Moreover, the training process is extremely expensive and suffers from local optima. This leads us to consider a novel approach of using *variable-order* HMMs via data mining. The basic idea is to use data mining to mine frequent patterns, which may be of different lengths, and then use these frequent patterns as the estimates of the joint probabilities, which can be used to seed the variable-order HMM. There has been almost no work on variable-order HMMs. The closest work is that of [58], who proposed using frequent associations for support, confidence and error pruned markov models (S/C/E-PMMs). However, we plan to use other higher order patterns (sequences) [89].

A. Estimation of HMM parameters

The parameters estimation problem can be divided into two categories:

1) **Structure of the HMM:** Define the number of states N, how they are connected, and the number, M, of output symbols in each state.
2) **Parameters value estimation:** Estimate the transition probability matrix A, the emission probabilities B, and the initial probabilities Π.

For both categories we will assume that the data at hand is composed of example sequences (training sequence), denoted as $O = \{o_1, o_2 \cdots o_T\}$, that are independent. Thus:

$$P(O|\lambda) = \sum_{t=1}^{T} P(o_t|\lambda) \tag{1}$$

Due to the complexity of the problem and the finite number of observations, there is no known analytical method so far for estimating λ to globally maximize $P(O|\lambda)$. Instead, iterative methods that provide a local maxima on $P(O|\lambda)$ can be used such as the frequently used Baum-Welch estimation algorithm [13]. Besides the well known Viterbi and Baum-Welch methods [67], in [70], the authors used a gradient descent method to estimate the HMM parameters, and a back-propagation neural network to determine the states of the HMM. In [58], associative mining was used to estimate the parameters of an all K^{th}-order Prediction-by-Partial-Match (PPM) Markov Predictors. We will look more closely at this method later on and compare it to the method we present in this work.

B. Baum-Welch Algorithm

We will describe in more detail the Baum-Weltch algorithm [10]–[14], since we use it as a base against which we compare our method. Recall that we want to estimate $\lambda = (A, B, \Pi)$ to maximize $P(O|\lambda)$. An overview of the iterative Baum-Welch algorithm is described in Algorithm 4.

Algorithm 4 Baum-Welch Algorithm

```
procedure FORWARD-BACKWARD($I$, $min\_conf$)
    Start with an initial model $\lambda_0$
    Compute new $\lambda$ based on $\lambda_0$ and the observation sequence $O = o_1, \cdots o_T$
    if $\log P(O|\lambda) - \log P(O|\lambda_0) < \Delta$ (where $\Delta$ is a predefined threshold) then
        stop
    else
        set $\lambda_0 \leftarrow \lambda$ and goto step 2
    end if
end procedure
```

The training mechanism, step 2, of the Baum-Welch algorithm uses the Forward-Backward algorithm [10] to compute the expected number of times each transition or emission is used, given the training sequence O. This computation is in the order $O(N^2T)$. The λ, Π , A and B values are iteratively computed, and converge in a continuous space until a converging criteria is met, typically stopping when the difference in the log of likelihood $[\log(P(O|\lambda)) - \log(P(O|\lambda_0))]$ is smaller than a predefined threshold Δ or the maximum number of iterations is reached. At each iteration, the log likelihood of the model λ is increased, converging the model to a local maximum. As pointed out in the beginning of this section, in a continuous-valued space there is no known method to get a global optimum, but rather a local maximum can be reached. The values of the initial parameters, used in the Baum-Welch algorithm, influence heavily the local maximum that the model converges to. This becomes a severe problem when dealing with large scale HMMs, which is the case in most of real-world applications, if not all of them.

TABLE X

SUBSEQUENCES OF LENGTH 1 MINED BY VGS

Index	Element	*freq*
1	C	2
2	D	2
3	H	1
4	I	1
5	E	1
6	F	1
7	G	1
8	A	4
9	B	3

C. Modeling: Variable-Order State Machine

VOGUE uses the mined sequences to build a variable order/gap state machine. The main idea here is to model each *non-gap* symbol in the mined sequences as a state that emits only that symbol and to add intermediate gap states between any two non-gap states. The gap states will capture the distribution of the gap symbols and length. Let F be the set of frequent subsequences mined by VGS, and let k be the maximum length of any sequence. While VOGUE can be generalized to use any value of $k \geq 2$, for clarity of exposition we will illustrate the working of VOGUE using mined sequences of length $k = 2$. We consider the general case in the next section. Let F_1 and F_2 be the sets of all frequent sequences of length 1 and 2, respectively, so that $F = F_1 \cup F_2$. Thus, each mined sequence $s_i \in F_2$ is of the form $s_i : v_f \to v_s$, where $v_f, v_s \in \Sigma$. Let $\Gamma = \{v_f | v_f \to v_s \in F_2\}$ be the set of all the distinct symbols in the first position, and $\Theta = \{v_s | v_f \to v_s \in F_2\}$ be the set of all the distinct symbols in the second position, across all the mined sequences $s_i \in F_2$. The VOGUE model is specified by the 6-tuple $\lambda = \{Q, \Sigma, A, B, \rho, \pi\}$ where each component is defined below:

Alphabet (Σ): The alphabet for VOGUE is given as:

$$\Sigma = \{v_1, \cdots v_M\} \tag{2}$$

where $|\Sigma| = M$ is the number of observations emitted by any state. The alphabet's size is defined by the number of symbols that occur at least once in the training data, obtained as a result of the first iteration of VGS, as shown in Table X. For our example S in Section 2.3 of Chapter 2, we have nine distinct frequent symbols, thus $M = 9$.

Set of States (Q): The set of states in VOGUE is given as:

$$Q = \{q_1, \cdots, q_N\}, \tag{3}$$

where:

$$|Q| = N = N_f + G_i + N_s + G_u. \tag{4}$$

Here, $N_f = |\Gamma|$ and $N_s = |\Theta|$ are the number of distinct symbols in the first and second positions, respectively. Each frequent sequence $s_i \in F_2$ having a gap $g \geq 1$ requires a gap state to models the gaps. G_i thus gives the number of gap states required. Finally $G_u = 1$ corresponds to an extra gap state, called *universal gap*, that acts as the default state when no other state satisfies an input sequence. For convenience let the partition of Q be:

$$Q = Q_f \cup Q_i \cup Q_s \cup Q_u \tag{5}$$

where the first N_f states belong to Q_f, the next G_i states belong to Q_i, and so on.

For our example S in Section 2.3 of Chapter 2, we have $N_f = 4$, since there are four distinct starting symbols in Table VIII (namely, A, B, C, D). We also have four ending symbols, giving $N_s = 4$. The number of gap states is the number of sequences of length 2 with at least one occurrence with gap $g \geq 1$. Thus $G_i = 8$, $C \to B$ is the only sequence that has all consecutive ($g = 0$) occurrences. With one universal gap state $G_u = 1$, our model yields $N = 4+8+4+1 = 17$ states.

Transition Probability Matrix (A): The transition probability matrix between the states:

$$A = \{a(q_i, q_j) | 1 \leq i, j \leq N\} \tag{6}$$

where:

$$a(q_i, q_j) = P(q^{t+1} = q_j | q^t = q_i) \tag{7}$$

gives the probability of moving from state q_i to q_j (where t is the current position in the sequence). The probabilities depend on the types of states involved in the transitions. The basic intuition is to allow transitions from the first symbol states to either the gap states or the second symbol states. The second symbol states can go back to either the first symbol states or to the universal gap state. Finally the universal gap state can go to any of the starting states or the intermediate gap states. We discuss these cases below.

- ***Transitions from First States***: Any first symbol state $q_i \in Q_f$ may transition to either a second symbol state $q_j \in Q_s$ (modeling a gap of $g = 0$) or to a gap state $q_j \in Q_i$ (modeling a gap of $g \in [1, maxgap]$). Let $s_{iy} : v_i \to v_y \in F_2$ be a subsequence mined by VGS. Let $freq_i^g(y)$ denote the frequency of s_{iy} for a given gap value g, and let $freq_i(y)$ denote the total frequency of the sequence, i.e.:

$$freq_i(y) = \sum_{g=0}^{maxgap} freq_i^g(y) \tag{8}$$

Let the fraction of gap-less transitions from q_i to q_j over all the transitions from q_i to $q_y \in Q_s$ be denoted as:

$$R = \frac{freq_i^0(j)}{\sum\limits_{y \in Q_s} freq_i(y)} \tag{9}$$

The transition probabilities from $q_i \in Q_f$ are given as:

$$a(q_i, q_j) = \begin{cases} R, & q_j \in Q_s \\ \dfrac{freq_i(j)}{\sum\limits_{y \in Q_s} freq_i(y)} - R, & q_j \in Q_i \\ 0, & q_j \in Q_f \cup Q_u \end{cases} \tag{10}$$

- ***Transitions from Gap States***: Any gap state $q_i \in Q_i$ may only transition to second symbol state $q_j \in Q_s$. For $q_i \in Q_i$ we have:

$$a(q_i, q_j) = \begin{cases} 1, & q_j \in Q_s \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

- ***Transitions from Second States***: A second symbol state $q_i \in Q_s$ may transition to either first symbol state $q_j \in Q_f$ (modeling a gap of $g = 0$), or to the universal gap state $q_j \in Q_u$ (modeling other gaps). Let $T = \sum_{s_x \in F_2} freq(s_x)$ be the sum of frequencies of all the sequences in F_2. For $q_i \in Q_s$ we have:

$$a(q_i, q_j) = \begin{cases} 0.99 \times \dfrac{\sum\limits_{q_y \in Q_f} freq_j(y)}{T}, & q_j \in Q_f \\ 0.01, & q_j \in Q_u \\ 0, & q_j \in Q_i \cup Q_s \end{cases} \tag{12}$$

Transitions back to first states are independent of q_i, i.e., the same for all $q_i \in Q_s$. In fact, these transitions are the same as the initialization probabilities described below. They allow the model to loop back after modeling a frequent sequence. Note, that we are primarily modeling the frequent sequences mined by VGS. However, we need to account for symbols that may appear between the frequent sequences but are not picked up as frequent. In a statistical model such as HMM, if a position or a subsequence of positions in an observation sequence is not present, then the probability of the sequence with respect to the model will be very small, regardless of how well it may match the rest of the model.

For example, assume a model built on two frequent subsequences $A \to B$ and $C \to D$. The sequence $S' = \mathbf{AB}R\mathbf{CD}$ should be clearly identified to be a good match to the model since both subsequences $A \to B$ and $C \to D$ are present in it. However, after being in the state that will generate the symbol B we need to generate the symbol R that is not in any of the two frequent sequences. Therefore, we need a transition to the universal gap state. However, since $R \to B$ was not identified as being frequent by VGS we need to assign a small probability to this transition. Therefore, we assign an empirically chosen value of 1% to the transition from the "second states" to the universal gap state. Furthermore, since $\sum_{j=1}^{N} a(q_i, q_j) = 1$ we assign the remaining 99% to the transition to the "first states".

- ***Transitions from Universal Gap*:** The universal gap state can only transition to the first states or the intermediate gap states. For $q_i \in Q_u$ we have:

$$a(q_i, q_j) = \begin{cases} 0.9 \times \dfrac{\sum_{q_y \in Q_f} freq_j(y)}{T}, & q_j \in Q_f \\ 0.1 \times \dfrac{1}{G_i}, & q_j \in Q_i \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

Since the first states can emit only one symbol, we allow transitions from universal gap to intermediate gap states, to allow for other symbol emissions. For example, assuming, as before, the same frequent sequences are $A \to B$ and $C \to D$ that are used to build the model. If we have a new observation sequence $S'' = \mathbf{AB}RF\mathbf{CD}$, clearly we need to generate two symbols, R and F, between the two frequent sequences $A \to B$ and $C \to D$. After generating B from a second state, we can generate R from the universal gap state however we need to generate one more symbol, F. F does not belong to any of the frequent sequence before generating the other frequent sequence $C \to D$. Therefore there is a need to transit from the gap state to an intermediate gap state since the first and second symbols can only generate symbols that belong to the frequent sequences. Moreover, since generating F after R was not identified as frequent by VGS, we need to assign a small probability to the transition from the universal gap to the intermediate gap states. This probability is at most 10% (empirically chosen) across all the gap states. In the remaining 90% cases, the universal gap transitions to a first state with probabilities proportional to its frequency.

Figure 2 shows transitions between states and their probabilities in VOGUE for our running example. Note, that each gap state's duration is considered explicitly within a state. The notation g_i, for example g_3, in the graph is the name of the gap state between the elements of the sequence, in this case $C \to D$, and not the value of the gap. The symbol states, on the other hand, are named after the *only* symbol that can be emitted from them, for example C is the *only* symbol that is emitted from the first symbol state.

Symbol Emission Probabilities (B): The symbol emission probabilities are state specific. We assume that each non-gap state ($q_i \in Q_f \cup Q_s$) outputs only a single symbol, whereas gap states ($q_i \in Q_i \cup Q_u$) may output different symbols. The emission probability matrix is then given as:

$$B = \{b(q_i, v_m) = P(v_m|q_i), 1 \le i \le N \text{ and } 1 \le m \le M\} \tag{14}$$

where:

$$b(q_i, v_m) = P(v_m|q_i) \tag{15}$$

is the probability of emitting symbol v_m in state q_i. $b(q_i, v_m)$ differs depending on whether q_i is a gap state or not. Since there is a chance that some symbols that do not occur in the training data may in fact be present in the testing data, we assign them a very small probability of emission in the gap states.

- ***Non-gap States:*** the emission probability is:

$$b(q_i, v_m) = \begin{cases} 1, & q_i \in Q_f \cup Q_s \\ 0, & otherwise \end{cases} \tag{16}$$

  For example, the first and second states are labeled by their emission symbol in Figure 2.
- ***Universal Gap:*** For $q_i \in Q_u$ we have:

$$b(q_i, v_m) = \left(\frac{freq_i(v_m)}{\sum_{v_m \in \Sigma} freq_i(v_m)} \right) \times 0.99 + c' \tag{17}$$

  where $c' = \frac{0.01}{M}$.
  This means that v_m is emitted with probability proportional to its frequency in the training data. The c' term handles the case when v_m does not appear in the training set.
- ***Gap States:*** If $q_i \in Q_i$ its emission probability depends on the symbol distribution mined by VGS. Let Σ_{q_i} be the set of symbols that were observed by VGS in the gap q_i. We have:

$$b(q_i, v_m) = \left(\frac{\sum_{g \ge 1} freq_g(v_m, q_i)}{\sum_{v_m \in \Sigma_{q_i}} \sum_{g \ge 1} freq_g(v_m, q_i)} \right) \times 0.99 + c \tag{18}$$

  where $c = \frac{0.01}{|\Sigma_{q_i}|}$.

Note that the above summations are for gap ranges $g \in [1, maxgap]$, since gap $g = 0$ is treated as a direct transition from one state to another. Note that the values 0.99 and 0.01 above arise from the pseudo-count approach used for previously unseen symbols.

In our running example, for the symbol $v_m = C$ and the gap state g_4 between the states that emit A and B, we have the frequency of C as 2 out of the total number (5) of symbols seen in the gaps (see Section 2.3 of Chapter 2). Thus C's emission probability is $\frac{2}{5} \times 0.99 + \frac{0.01}{4} = 0.385$.

Gap Duration Probabilities (ρ): The probability of generating a given number of gaps from the gap states Q_i is given by the gap duration probability matrix:

$$\rho = \{\rho(q_i, g) | q_i \in Q_i, g \in [1, maxgap]\} \tag{19}$$

Let q_i be the gap state between a state $q_x \in Q_f$ and a state $q_y \in Q_s$ corresponding to the sequence $s : v_i \to v_y \in F_2$. The gap duration probability is proportional to the frequency of observing a given gap value for s, i.e.:

$$\rho(q_i, g) = \begin{cases} \dfrac{freq_i^g(y)}{\sum_{g \in \{1, \cdots, maxgap\}} freq_i^g(y)}, & q_i \in Q_i \\ 1, & q_i \in Q \backslash Q_i. \end{cases} \tag{20}$$

In our running example, for the gap state g_4 between the states that emit A and B, we have $\rho(g_4, 2) = \frac{2}{3} = 0.67$, since we

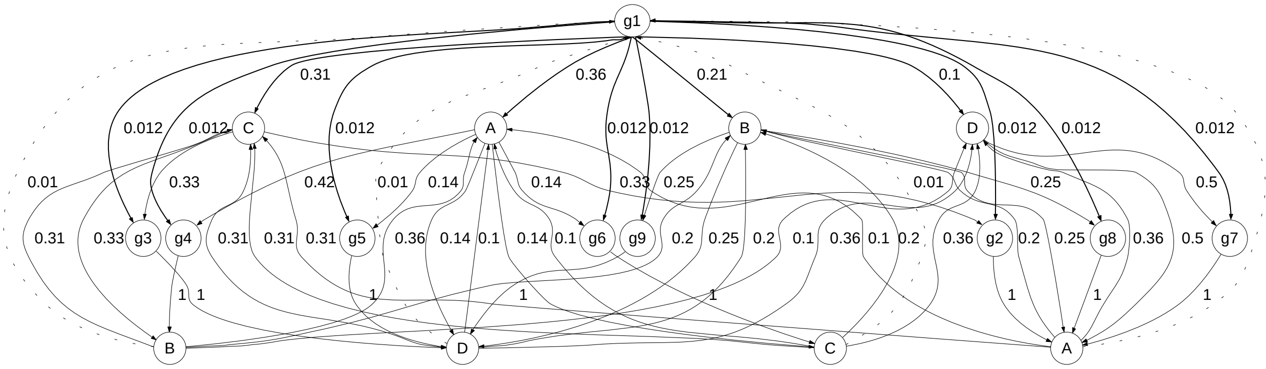


Fig. 2. VOGUE State Machine for Running Example

observe twice a gap of 2, out of three occurrences.

Initial State Probabilities (π): The probability of being in state q_i initially is given by $\pi = \{\pi(i) = P(q_i|t = 0), 1 \leq i \leq N\}$, where:

$$\pi(i) = \begin{cases} 0.99 \times \dfrac{\displaystyle\sum_{q_y \in Q_f} freq_i(y)}{T}, & q_i \in Q_f \\ 0.01, & q_i \in Q_u \\ 0, & q_i \in Q_i \cup Q_s \end{cases} \tag{21}$$

We use a small value for the Universal Gap state as opposed to the states in Q_f to accentuate the patterns retained by VGS while still providing a possibility for gaps after and before them.

Note that the values 0.99, 0.1, 0.01, etc., used in the transition and emission probabilities, are obtained by using pseudo-counts [17] to allow for symbols that are unseen in the training data set.

D. Generalization of VOGUE to $k \geq 2$

Here, we generalize the model to $k \geq 2$. Let S be the set of subsequences mined by VGS, and let k be the maximum length of any sequence. We denote by k_seq a sequence of length k, such as $v_{i_1}, v_{i_2}, \cdots, v_{i_k}$. Let $\Gamma_k(j)$ be the set of symbols in the j^{th} position in all subsequences s, $s \in S$, of length up to k, $j = 1, \cdots, k-1$; $\Gamma_k(k)$ is then the set of different last symbols in all subsequences of length up to k. Finally, let $S(j)$ be the set of subsequences in S of length at least j.

The VOGUE's state machine is denoted as λ and is made up of the 6-tuple $\lambda = \{Q, V, A, B, \rho, \pi\}$ where each component is defined as follows:

- $Q = \{q_1, \cdots, q_N\}$ – the set of states of VOGUE, where N is the number of states of VOGUE such that: $N = N_1 + G_1 + \cdots + N_{i-1} + G_{i-1} + \cdots + N_k + G_k$, where:
  - $N_i = \mid \Gamma_k(i) \mid$, $i = 2, \cdots, k-1$. This denotes the number of distinct symbols in position i over all the sequences. Thus N_1 is the number of distinct first symbols and N_k is the number of distinct last symbols.
  - G_i (for $i < k$) is the number of distinct pairs of symbols in positions $i-1$ and i. This corresponds to the number of gap states required between states at positions $i-1$ and i.
  - $G_k = 1$, corresponds to an extra gap state, called Universal Gap state, that will capture elements not captured by any of the above.

  For convenience we let:
  - $\sum(G_j) = \sum_{l<j}(N_l + G_l)$ for all $j \in \{2, \cdots, k\}$ and $\sum(G_1) = 0$.
  - $\sum(N_j) = \sum_{l<j}(G_j) + N_j$ for all $j \in \{1, \cdots, k\}$.

  The states of VOGUE are, then, given as follows:
  - For $\sum(N_j) < i \leq \sum(G_{j+1})$, q_i corresponds to the gap of variable length between the $(j-1)^{th}$ and j^{th} elements in the subsequences, $j \in \{1, \cdots, k-1\}$. These states will be called "Gap" states.
  - For $\sum(G_j) < i \leq \sum(N_j)$, q_i corresponds to the elements in $\Gamma_k(j)$, $j \in \{1, \cdots, k\}$. These states will be called "Symbol" states.
  - For $i = \sum(G_{k+1})$, q_i corresponds to the Universal Gap state.
- $V = \{v_1, \cdots v_M\}$ – the alphabet of the symbols, where M is the number of observations emitted by a state in Q. It is the number of different subsequences of length 1 retained by VGS, unless stated differently by the application at hand.
- $A = \{a_{il}\}$ – the transition probability matrix between the states in Q. For convenience we let:
  - $R_{sj}(m)$: frequency of subsequence s, $s \in S$, where m is the index of the symbol $v_m \in \Gamma_k(j)$ at the j^{th} position of s.
  - $W_{jg}(m, m')$: frequency of subsequence s, $s \in S$, where m and m' are, respectively, the indexes of v_m and $v_{m'}$. v_m is at the j^{th} position of s and $v_{m'}$ is at the $j + 1^{th}$ position of s. g is the value of the gap between the j^{th} and $j + 1^{th}$ positions of s, $g \in \{1, \cdots, maxgap\}$.

  Each element of the matrix is given as $a_{il} = P(q_{t+1} = l|q_t = i)$, for $1 \leq i, l \leq N$.
  - if $\sum(G_j) < i \leq \sum(N_j)$, $j \in \{1, \cdots, k-1\}$, (Non-Gap (Symbol) states):

$$a_{il} = \begin{cases} \frac{\sum_{g=1}^{maxgap} W_{jg}(m,m')}{\sum_{s \in S} R_{sj}(m)}, & \text{if } \sum(N_j < l \leq \sum(G_{j+1})); \\ \frac{W_{j0}(m,m')}{\sum_{s \in S} R_{sj}(m)}, & \text{if } \sum(G_{j+1}) < l \leq \sum(N_{j+1}); \\ 0, & \text{Otherwise;} \end{cases} \tag{22}$$

  where m and m' are the indexes, respectively, of symbols v_m and $v_{m'} \in V$ such that $\exists v_{l'} \in \Gamma_k(j)$ and $\exists v_{l"} \in \Gamma_k(j+1)$: ($v_{l'} = v_m$ and $v_{l"} = v_{m'}$), and ($l' = i - \sum(G_j)$ and $l" = l - \sum(G_{j+1})$).
  - if $\sum(N_j) < i \leq \sum(G_{j+1})$, $j \in \{1, \cdots, k-1\}$, (Gap states):

$$a_{il} = \begin{cases} 0.9, & \text{if } \sum(G_{i+1}) < l \leq \sum(N_{i+1}); \\ \frac{0.1}{N_1+1} & \text{if } \sum(N_k) < l \leq \sum(G_{k+1}) \\ \frac{0.1}{N_1+1} & \text{if } \sum(G_1) < l \leq \sum(N_1) \\ 0, & \text{Otherwise;} \end{cases} \tag{23}$$

  - if $\sum(G_k) < i \leq \sum(N_k)$ (last Symbol states):

$$a_{il} = \begin{cases} \left(\frac{\sum_{s\in S} R_{sj}(m_i)}{\sum_{m\in\Gamma_k(j)}\sum_{s\in S} R_{sj}(m)}\right) \times 0.9, \\ \quad \text{if } \sum(G_1) < l \leq \sum(N_1); \\ 0.1, \text{if } l = \sum(G_{k+1}); \\ 0, \text{Otherwise}; \end{cases} \tag{24}$$

where $R_{sj}(m_i)$ is as defined earlier and m_i is the index of symbol $v_{m_i} \in V$ such that $\exists v_{l'} \in \Gamma_k(j)$: $v_{l'} = v_{m_i}$, and $l' = i - \sum(G_j)$.

- if $i = \sum(G_{k+1})$ (Universal Gap state):

$$a_{il} = \begin{cases} \left(\frac{\sum_{s\in S} R_{sj}(m_i)}{\sum_{m\in\Gamma_k(j)}\sum_{s\in S} R_{sj}(m)}\right) \times 0.9, \\ \quad \text{if } \sum(G_1) < l \leq \sum(N_1); \\ \frac{0.1}{G}, \text{if } \sum(N_j) < l \leq \sum(G_{j+1}),\ j \neq k; \\ \\ 0, \text{Otherwise}; \end{cases} \tag{25}$$

Note: Since each gap state's duration is considered explicitly within a state, there is no self-transition to any state. The state transition probabilities to the same state is, then, $a_{ii}, 1 \leq i \leq N$, are set to 0.

- $B = \{b_i(m) = P(o_t = v_m | q_t = i), 1 \leq i \leq N$ and $1 \leq m \leq M\}$ - the emission probability of state i. It is defined as follows:
  - if $\sum(G_j) < i \leq \sum(N_j)$, $j \in [1, k]$ (Non-Gap (Symbol) states):

$$b_i(m) = \begin{cases} 1, \text{if } \exists v_l \in \Gamma_k(j) \text{ s.t: } v_l = v_m \text{ and} \\ \quad l = i - \sum(G_j); \\ 0, \text{otherwise}; \end{cases} \tag{26}$$

  - if $\sum(N_j) < i \leq \sum(G_{j+1})$, $j \in [1, k]$ (Gap states):

$$b_i(m) = \left(\frac{\sum_{g=1}^{MAXGAP} freq_g(m,l)}{\sum_{v_m\in\Psi_j(s)}[\sum_{g=1}^{MAXGAP} freq_g(m,l)]}\right) \times 0.99 + c \tag{27}$$

where $freq_g(m, l)$ is the frequency of v_m such that $v_m \in \Psi_j(s_l)$, $s_l \in S(j)$ and $l = i - \sum(G_j)$; $c = \frac{0.01}{G_j}$.

  - if $\sum(N_k) < i \leq \sum(G_{k+1})$, (Universal Gap state):

$$b_i(m) = \left(\frac{freq_i(m)}{\sum_{v_m\in V} freq_i(m)}\right) \times 0.99 + c' \tag{28}$$

where $freq(m)$ is the frequency of v_m, $v_m \in V$ and v_m is also the 1-seq retained by VGS; $c' = \frac{0.01}{M}$.

Note: We will consider that each state, except *gap* states, generates either only one symbol from the alphabet *V* at all times or generates an element from a subclass of symbols of the alphabet.

- $\rho = \{\rho_{ig} =, 1 \leq i \leq N, 1 \leq g \leq maxgap\}$ - the gap states duration probability matrix:
  - if $\sum(G_j) < i \leq \sum(N_j)$, $j \in [1, k]$ (Non-Gap (Symbol) states) and $\sum(N_k) < i \leq \sum(G_{k+1})$ (Universal Gap state):

$$\rho_{ig} = \begin{cases} 1, & \text{if } g = 0; \\ 0, & \text{otherwise}; \end{cases} \tag{29}$$

  - if $\sum(N_j) < i \leq \sum(G_{j+1})$, $j \in [1, k]$ (Gap states):

$$\rho_{ig} = freqgap_l(g) \sum_{g=1}^{M} freqgap_l(g) \tag{30}$$

where $freqgap_l(g)$ is the frequency of s_l, $s_l \in S(j)$ and $l = i - \sum(G_j)$, such that the gap between the j^{th} and the $(j+1)^{th}$ elements is equal to "g".

TABLE XI

VGS WITH $k = 2$, $maxgap = 2$ AND $min\_sup = 2$

Subsequence	*freq*	$g = 0$	$g = 1$	$g = 2$
$A \to B$	4	1	1	2
$B \to A$	3	1	1	1

- $\pi = \{\pi(i) = P(q_0 = i), 1 \leq i \leq N\}$- the initial probabilities are estimated as follows:

$$\pi(i) = \begin{cases} \left(\frac{\sum_{s\in S} R_{sj}(m_i)}{\sum_{m\in\Gamma_k(j)}\sum_{s\in S} R_{sj}(m)}\right) \times 0.99, \\ \quad \text{if } \sum(G_j) \leq i \leq \sum(N_j),\ j = 1; \\ 0.01, \text{if } i = \sum(G_{k+1}); \\ 0, \text{otherwise }; \end{cases} \tag{31}$$

where $R_{sj}(m)$ is as defined earlier and m_i is the index of symbol $v_{m_i} \in V$ such that $\exists v_{l'} \in \Gamma_k(j)$: $v_{l'} = v_{m_i}$, and $l' = i - \sum(G_j)$..

V. VOGUE VARIATIONS

Here, we describe two variations of VOGUE, namely, **C**anonical VOGUE (**C-VOGUE**), and **K**nowledge VOGUE (**K-VOGUE**). C-VOGUE intends to decrease even more the state space complexity by pruning frequent sequences mined by VGS that are artifacts of other "primary" frequent sequences, thus, pruning the states and transitions from and to those states. Therefore further decreases the state space complexity. On the other hand, in some domains, such as in biological sequence analysis, the patterns that are supposed to be frequent do not have an exact match in all the sequences in the original data set. In fact, some elements of these patterns could be different, however, they share some common characteristics and thus called "*similar*". Therefore, the sequences in the data set that have a sequence of elements that are not similar but share some common characteristics could be incorporated into the model. K-VOGUE is an extension of VOGUE that takes into consideration these constraints.

A. C-VOGUE: Canonical VOGUE

Some patterns mined by VGS are artifacts of other patterns. For example, let us consider the sequence S = **ABA**C**B**D**A**EF**B**GH**A**IJ**B**. Using VGS with $maxgap = 2$, $min\_sup = 2$ and $k = 2$, the frequent subsequences are shown in Table XI. For the frequent subsequence $A \to B$, its frequency is *freq* $= 4$:

- Once with a gap of length 0: that is (AB).
- Once with a gap of length 1: that is the subsequence (ACB) where A is followed by B but with one element in between, namely C.
- Twice with a gap of length 2: that is the subsequences $(AEFB)$ where A is followed by B with two elements, E and F, between them, and the subsequence $(AIJB)$ where A is followed by B with two elements, I and J, in between.

On the other hand, the frequent sequence $B \to A$ was mined by VGS as being frequent under the constraints $maxgap = 2$, $min\_sup = 2$ and $k = 2$. In fact, its frequency is *freq* $= 3$:

- Once with a gap of length 0: that is the subsequence (BA) where B is followed directly by A.
- Once with gap length 1: that is the subsequence (BDA) where B is followed by A with one element, D, between them.

TABLE XII

THE *eid* OF THE DIFFERENT ITEMS IN S

item	(Sid, Eid)
A	$(1,1)$, $(1,3)$, $(1,7)$, $(1,13)$
B	$(1,2)$, $(1,5)$, $(1,10)$, $(1,16)$
C	$(1,4)$
D	$(1,6)$
E	$(1,8)$
F	$(1,9)$
G	$(1,11)$
H	$(1,12)$
I	$(1,14)$
J	$(1,15)$

- Once with gap length 2, that is the subsequence $(BGHA)$ where B is followed by A with two elements, G and H in between.

This frequent subsequence seems to be legitimate under VGS constraints, however, all the elements, B and A, involved in this sequence are already in the frequent subsequence $A \rightarrow B$. Indeed, this makes $B \rightarrow A$ not a new pattern but an "*artifact*" pattern of $A \rightarrow B$ since $A \rightarrow B$ is more frequent than $B \rightarrow A$. Moreover, the position of the items involved in $B \rightarrow A$ are the same as those of $A \rightarrow B$. For instance, in the subsequence (AB), one of the subsequences in the count of the frequent sequence $A \rightarrow B$, the *position* or *eid* of A is 1 and the *eid* of B is 2. While for the frequent sequence $B \rightarrow A$, in the subsequence (BA) the *eid* of B is 2 and the the *eid* of A is 3. Therefore, the B that is at the *eid* 2 is the same and hence *shared* between the two subsequence (AB) and (BA). Table XII shows the items in S and their *eid*s.

The items in S that are *shared* between $A \rightarrow B$ and $B \rightarrow A$ in the format of $(items, eid)$ are $\{(B,2),(A,3),(B,5),(A,7),(B,10),(A,13)\}$. Therefore, the subsequence (BA) is an *artifact* of (AB) and (ACB). The subsequence (BDA) is an artifact of (ACB) and $(AEFB)$. The subsequence $(BGHA)$ is an artifact of $(AEFB)$ and $(AIJB)$. Hence, the frequent subsequence $B \rightarrow A$ is an "*artifact*" pattern of the "*primary*" pattern $A \rightarrow B$.

Figure 3 shows the structure of VOGUE State Machine with $N = 7$ being the number of states for our example sequence. In fact, $N_f = 2$ since there are 2 distinct starting symbols in Table VII (namely A and B). We also have 2 ending symbols giving (namely A and B)$N_s = 2$. The number of gap states is the number of sequences of length 2 that has at least one occurrence with gap $g \geq 1$, thus, $G_i = 2$. Since we have one universal gap $G_u = 1$, our model yields $N = N_f + G_i + N_s + G_u = 2 + 2 + 2 + 1 = 7$. We can see clearly that the paths $A \rightarrow B$ and $B \rightarrow A$ can be merged into one path $A \rightarrow B$ and the information of gap g_2 and g_3 can be also contained in one gap state instead of two. In fact, for our example, by eliminating $B \rightarrow A$, and hence the states associated with the path $B \rightarrow A$ from the frequent mined sequences to build the Variable-Order state machine the number of states drops from $N = 7$ to $N = 4$. By doing so, we reduce the state space complexity significantly while conserving the coverage and accuracy as we will show through experimental results in Chapter 6.

After showing the benefits of pruning the states associated with the *artifact* patterns, we need a special pruning mechanism to separate primary patterns from artifact patterns.

We define the constraint that distinguish the "*primary*" patterns from the artifact as follows:

Definition: Let $\zeta_p = \{(e, eid) | e$ in $S_p\}$ be the set of pairs, (e, eid), of elements e in the "*primary*" S_p and their corresponding eid in the original sequence S. Let $\zeta_a = \{(e, eid) | e$ in $S_a\}$ be the set of pairs, (e, eid), of elements e and their corresponding eid in the artifact pattern S_a. Therefore, the third constraint is that for a candidate sequence to be a "*artifact*" pattern to a "*primary*" pattern the following conditions have to be satisfied as well: (i) $\zeta_a \subset \zeta_p$ and (ii) $\exists$ $(e', eid'), (e'', eid'') \in \zeta_p$ such that $eid' < eid < eid''$, where $(e, eid) \in \zeta_a$. This constraint is necessary. As consequence of this constraint we have the following condition:

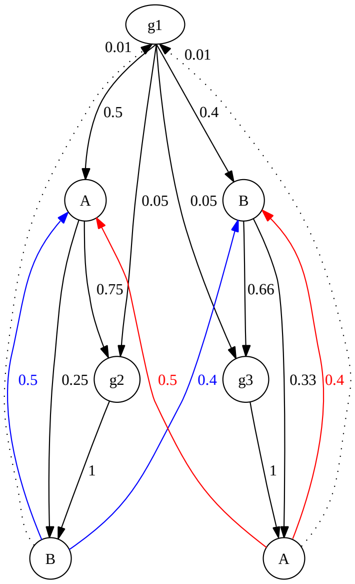


Fig. 3. VOGUE State Machine before pruning the artifact "$B \rightarrow A$" for the example sequence S.

TABLE XIII

VGS WITH $k = 2$, $maxgap = 2$ AND $min\_sup = 2$ FOR SEQUENCE S'

Subsequence	*freq*	$g = 0$	$g = 1$	$g = 2$
$A \rightarrow B$	3	1	1	1
$B \rightarrow A$	2	1	0	1

Condition 1: The artifact patterns are less frequent than the corresponding primary ones. In fact, for our running example, the frequency of the *primary* pattern $A \rightarrow B$ is *freq* $= 4$, while the frequency of the *artifact* pattern $B \rightarrow A$ is *freq* $= 3$.

In the case of when the mined patterns by VGS are of length $k = 2$, the following condition is a consequence of the constraint of artifact patterns:

Condition 2:The elements of the primary and the artifact patterns are "*mirrored*". In fact, let $\alpha \rightarrow \beta$ be the primary pattern and $\kappa \rightarrow \nu$ be the artifact pattern. On one hand, the first element, κ, in the artifact pattern is the same as the second last element, β, in the corresponding primary pattern ($\kappa = \beta$). On the other hand, the last element,ν, of the artifact pattern is the same as the first element, α, in the corresponding primary pattern ($\alpha = \nu$). In fact, in our running example, the first element of the artifact pattern $B \rightarrow A$ is B, which is the last element in the corresponding primary pattern $A \rightarrow B$. Likewise, the last element of the artifact pattern $B \rightarrow A$ is A, which is the first element of the primary pattern $A \rightarrow B$. Therefore, we consider that the artifact pattern $B \rightarrow A$ is a ***mirror*** of the primary pattern $A \rightarrow B$.

However, if these two conditions hold they are not sufficient. The necessary condition is the constraint defined earlier.In fact, let us assume that the data set in our example consists of the new sequence:

$$S' = \mathbf{AB}RST\mathbf{A}C\mathbf{B}VDW\mathbf{A}EF\mathbf{B}GH\mathbf{B}IJ\mathbf{A}KLM\mathbf{BA}.$$

Then the frequent sequences are $A \rightarrow B$ and $B \rightarrow A$ under the conditions $min\_sup = 2$, $maxgap = 2$ and $k = 2$, as shown in Table XIII.

The frequency of $A \rightarrow B$is *freq* $= 3$:

- Once with a gap of length 0: that is the subsequence (AB) where A is followed directly by B in positions 1 and 2 respectively in sequence S'.

- Once with gap length 1: that is the subsequence (ACB) where A is followed by B with one element, C, between them.
- Once with gap length 2, that is the subsequence $(AEFB)$ where A is followed by B with two elements, E and F in between.

While the frequency of $B \rightarrow A$is *freq* $= 2$:

- Once with a gap of length 0: that is the subsequence (BA) where B is followed directly by A in the positions 25, 26 respectively.
- Once with gap length 2: that is the subsequence $(BGHA)$ where B is followed by A with two elements, G and H, between them.

Although $B \rightarrow A$ is a ***mirror*** of $A \rightarrow B$ and the frequency of $B \rightarrow A$ is *freq* $= 2$ which is less than the one of $A \rightarrow B$ (*freq* $= 3$), $B \rightarrow A$ cannot be considered an artifacts pattern of $A \rightarrow B$. This is because the positions or *eid* of the elements A and B in $A \rightarrow B$ are different and far from those in $B \rightarrow A$. Therefore, $B \rightarrow A$ is not in the span of $A \rightarrow B$ or vice versa. Indeed, as shown in Table XIV, the elements A and B in $A \rightarrow B$ ($\zeta_p = \{(A,1),(A,6),(A,12),(B,2),(B,8),(B,15)\}$) all have *eid*s that are different and less in value than the ones in $B \rightarrow A$ ($\zeta_a = \{(A,21),(A,26),(B,18),(B,25)\}$). Note that we are considering patterns with a $maxgap = 2$ that is why $(A,21)$ $(B,25)$ are not in $\zeta_{A \rightarrow B}$ because in this case A is followed by B with 3 elements in between. Therefore, the two sequences are not interleaved. On the other hand, as shown in Table XII for the example sequence S, the set ζ_a of the pairs of elements and their *eid*s of $B \rightarrow A$ is:

$$\zeta_a = \{(A,3),(A,7),(A,13),(B,2),(B,5),(B,10)\}$$

and the one of $A \rightarrow B$ is:

$$\zeta_p = \{(A,1),(A,3),(A,7),(A,13),(B,2),(B,5),(B,10),(B,16)\}$$

. We see clearly that $\zeta_a \subset \zeta_p$. Moreover, B in $A \rightarrow B$ has an *eid* that is greater than any ones identified in ζ_a, that is $(B,16)$. Likewise, the element A has an extra *eid* in $A \rightarrow B$, which corresponds to $(A,1)$ the smallest *eid* value in S.

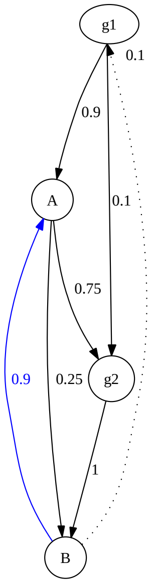


Fig. 4. C-VOGUE State Machine after pruning the artifact "$B \rightarrow A$" for the example sequence S.

Figure 4 shows the new Variable-Order state machine that results from this pruning. The path corresponding to the artifact pattern $B \rightarrow A$ was pruned. However, the path $B \rightarrow A$ can be still reproduced. In fact once in state n_2 that produces B, we can go directly to state n_1 that produces A. If there is a need to produce some elements between B and A, then there is a possibility to transit from state n_2 to the universal gap g_1, and produce an element or several elements, then transit to state n_1 to produce A.

TABLE XIV

THE *eid* OF THE DIFFERENT ITEMS IN S'

item	(*sid*, *eid*)
A	$(1,1)$, $(1,6)$, $(1,12)$, $(1,21)$, $(1,26)$
B	$(1,2)$, $(1,8)$, $(1,15)$, $(1,18)$, $(1,25)$
C	$(1,7)$
D	$(1,10)$
E	$(1,13)$
F	$(1,14)$
G	$(1,16)$
H	$(1,17)$
I	$(1,19)$
J	$(1,20)$
K	$(1,22)$
L	$(1,23)$
M	$(1,24)$
R	$(1,3)$
S	$(1,4)$
T	$(1,5)$
V	$(1,9)$
W	$(1,11)$

The pruning process is added between the *Pattern Extraction* and the *Data Modeling* steps in VOGUE. We call this extension of VOGUE, Canonical VOGUE (C-VOGUE). All the steps in C-VOGUE except the "*Artifact Pruning*" process, are the same as in VOGUE. The mined frequent sequences from VGS that are pruned satisfy the following constraints that summarize the observations discussed earlier:

1) The artifact sequence is a ***mirror*** of the "*primary*" sequence.
2) If *freq*$_a$ is the frequency of the artifact sequence and *freq*$_p$ the frequency of the primary sequence, then *freq*$_a$ < *freq*$_p$.
3) $\zeta_a \subset \zeta_p$, and $\exists\ (e', eid'), (e'', eid'') \in \zeta_p$ such that $eid' < eid < eid''$ where $(e, eid) \in \zeta_a$.

B. K-VOGUE: VOGUE Augmented with Domain Specific Knowledge

In some domains such as in biological sequence analysis [24], [26], the patterns that are supposed to be frequent do not have an exact match in all the sequences that belong to the same family or class, but instead we need to allow inexact matches among some elements. These elements, however, share some common characteristics and thus called "*similar*". In fact, in proteins, the motifs (patterns or frequent subsequences) that characterize a family or class of proteins is a pattern that is found in all the sequences where some elements could be different. In this case the pattern or motif looks more like a grammar. For example, given the motif P $=\{G[IVT][LVAC][LVAC][IVT]D[DE][FL][DNST]\}$, the subsequences: $(GICCIDEFD)$, $(GVCLIDEFD)$, $(GVCLIDEFD)$, $(GVVCIDEFD)$, and $(GVVCIDEFD)$ can interchange. $[IVT]$ means that either I, V, or T could be found. The elements that are grouped together are in general amino acids that have similar structure. For example, I and V belong to the same subgroup, that is **non-polar** and **hydrophobic** amino acids group [52].

Regular methods that look for exact matches in subsequences to declare them frequent will miss a pattern such as the motif P. Therefore, there is a need for methods that allow for substitutions among similar elements to capture such patterns/motifs. In bioinformatics, substitution matrices, [7], [27], have been used to estimate the rate at which each possible element in a sequence changes to another element over time. Substitution matrices are used in amino acid sequence alignment, [18], [57], [76], [80], where the similarity between sequences depends on the mutation rates as represented in the matrix. Aligned sequences are typically written with their elements (amino acids) in columns, and gaps are inserted so that elements with identical or similar characters are aligned in the successive columns.

Thanks to the adaptability of VOGUE, it can be extended to accommodate inexact matches. We will extend VOGUE to allow for domain knowledge specific information to be taken into consideration during the pattern extraction and the modeling process. The extension of VOGUE that we propose in this section to allow for ***substitutions*** among ***similar*** elements is called **K**nowledge **VOGUE** (***K-VOGUE***).

The main extensions made to VOGUE to allow for substitutions are as follows:

1) Get the clusters of symbols used in the data set from the domain expert. Otherwise, cluster the alphabet symbols into clusters $\mathcal{C}_i$ based on similarity depending on the problem domain at hand using the domain knowledge. The clustering method we propose is described in some more detail in Section V-C.
2) Replace each element in the data set that belongs to a cluster $\mathcal{C}_i$, by a symbol c_i that represents the cluster $\mathcal{C}_i$. Therefore, the symbol alphabet now becomes the set $\sum' = \{c_i, i\{1, \cdots, L\}\}$, where L is the number of clusters $\mathcal{C}_i$. Therefore the alphabet symbol is no longer the original symbol set $\sum$ but the new symbol alphabet $\sum'$.
3) Mine the data set for frequent subsequences using VGS according to a user defined min_sup, $maxgap$ and length k.
4) Modeling the mined patterns could be done in two ways:
   a) Build the variable-order gap state machine directly from the mined patterns by VGS with the symbol states Q_f and Q_s still emitting only one symbol with probability 1. In this case, the symbols are the representatives c_i of the clusters $\mathcal{C}_i$, $i \in \{1, \cdots, L\}$ and not the original alphabet symbol $\sum$. For example, let's consider the cluster that has been identified as $\mathcal{C}_1 = \{I, V, T\}$ and the representative symbol is $c_1 = I$. In this case, the symbol states Q_f and Q_s, in the variable-Order state machine, produce only one symbol. That symbol is the representative symbol of one of the clusters $\mathcal{C}_i$. In our example, one of the sates in Q_f emits I the representative of the cluster $\mathcal{C}_1$. Therefore, for the *Interpretation* step we need to replace the elements in the testing data set with the cluster representatives $c_i, i \in \{1, \cdots, L\}$ then use VG-Viterbi with the modified testing data set.
   b) Build a variable-order gap state machine that allows several emissions from the symbol states Q_f and Q_s. In order to build the variable-order state machine, we use the frequent sequences mined by VGS from the modified data set,but the only difference is that the state symbols Q_f and Q_s can emit any symbol, v_m, from the corresponding cluster $\mathcal{C}_i$, rather than emitting only one symbol from the modified symbol alphabet $\sum'$. In our running example, a state in Q_f can emit any symbol v_m from $\mathcal{C}_1 = \{I, V, T\}$. The emission probability in this case, is computed as follows:

$$b(q_i, v_m) = \frac{freq_i(v_m)}{\sum_{v_m \in \mathcal{C}_i} freq_i(v_m)} \tag{32}$$

   Note that only the symbols belonging to the cluster $\mathcal{C}_i$ will be emitted. The symbols from the alphabet $\sum$ that do not belong to the cluster $\mathcal{C}_i$ will be emitted with probability 0. For example, let's assume that the frequency of I, V, and T, were 6, 4, and 1 respectively. Then the emission probability from state q_1 for I is $b(q_1,\text{I}) = \frac{6}{6+4+1} = \frac{6}{11} = 0.54$, for V is $b(q_1, V) = 0.36$, and for T is $b(q_1, T) = 0.09$. On the other hand, for any symbol $v_m \in \sum \backslash \mathcal{C}_1$ its corresponding emission probability is $b(q_1, v_m) = 0$. In this case, the VG-Viterbi will be used directly on the original testing data set with no replacement.

C. Symbol clustering

This section describes the clustering method that we use for clustering the symbols that will be used in K-VOGUE. Sometimes the clusters of the symbols are available from the expert in the domain at hand and sometimes they are not. In the latter case where the information from the domain expert is not available, we can use a clustering method to get the cluster from some domain information. K-means is one of the most popular clustering algorithms in data mining. A major drawback to K-means is that it cannot separate clusters that are non-linearly separable in input space. In many real world problems where we need to cluster the symbols, the clusters could be non-linearly separable like in the case of amino acids. Therefore, K-means, as it is, will not be a good clustering algorithm. Two recent approaches have emerged for tracking the problem. One is kernel K-means, where, before clustering, points are mapped to a higher-dimensional feature space using a nonlinear function, and then kernel k-means partitions the points by linear separators in the new space. The second approach is spectral clustering algorithms, which use the eigenvectors of a similarity matrix to partition points into disjoint clusters, with points in the same cluster having high similarity and points in different clusters having low similarity. Spectral clustering has many applications in machine learning, exploratory data analysis, computer vision and speech processing [86], [91]. Most techniques explicitly or implicitly assume a metric or a similarity structure over the space of configurations, which is then used by clustering algorithms. The success of such algorithms depends heavily on the choice of the metric, but this choice is generally not treated as part of the learning problem. Thus, time-consuming manual feature selection is often a necessary precursor to the use of spectral methods. Several recent papers have considered ways to alleviate this burden by incorporating prior knowledge into the metric, either in the setting of K-means clustering [79], [84] or spectral clustering [86], [91]. Several algorithms have been proposed in the literature [46], [59], [73], each using the eigenvectors in slightly different ways. A popular objective function used in spectral clustering is to minimize the normalized cut [73]. The *k-way normalized cut* problem is considered in [87] to partition the data set into k clusters. We will describe in some more detail the k-way normalized cut spectral clustering since it is the one that we will be using as clustering technique. If we represent the data set to be clustered as a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices representing data points, $\mathcal{E}$ is the set of edges connecting the vertices indicating pair-wise similarity between the points. The definition of a "good" clustering is that points belonging to the same cluster should be highly similar while the points belonging to different clusters should be highly dissimilar. Then represent the similarity graph as a matrix A called edge similarity matrix, assumed to be nonnegative and symmetric. If $n = |\mathcal{E}|$, then A is an $n \times n$ matrix and A_{ij} is the edge weight between vertex i and j. The eigenvalues and eigenvectors of the matrix A provide global information about its structure. Let's consider v and u as two subsets of $\mathcal{V}$, a $link$ is defined as:

$$link(u, v) = \sum_{i \in u, j \in v} A(i, j) \tag{33}$$

Then the normalized link-ratio of u and v is:

$$normallinkratio(u, v) = \frac{links(u, v)}{links(u, \mathcal{V})} \tag{34}$$

The k-way normalized cut problem is to minimize the links in a cluster relative to the total "weight" of the cluster. For k-way partitioning of the vertices, solving the following problem is of interest:

$$\min \frac{1}{k} \sum_{j=1}^{k} normalinkratio(\mathcal{V}_j, \mathcal{V} \backslash \mathcal{V}_j). \tag{35}$$

This problem was relaxed in [87] by the following spectral relaxation: Let D be the diagonal matrix where $D_{ii} = \sum_j A_{ij}$. Therefore, the normalized cut criterion is equivalent to:

$$\max \frac{1}{k} trace(Z^T AZ) \tag{36}$$

where $Z = X(X^T DX)^{-1/2}$, and X is an $n \times k$ indicator matrix for the partitions and $Z^T DZ = I_k$.

If we define $\tilde{Z} = D^{1/2} Z$ and relaxing the constraint that X is an indicator matrix, then the problem becomes a maximization of the trace of $\tilde{Z} D^{-1/2} AD^{-1/2} \tilde{Z}$, where the constraints on $\tilde{Z}$ are relaxed such that $\tilde{Z}^T \tilde{Z} = I_k$. This can be solved in turn by setting the matrix $\tilde{Z}$ to be the top k eigenvectors of the matrix $D^{-1/2} AD^{-1/2}$. Therefore the clustering algorithm is described as follows:

1) Pre-processing: Construct the scaled adjacency matrix $A' = D^{-1/2} AD^{-1/2}$
2) Decomposition:
   - Find the eigenvalues and eigenvectors of A'.
   - Build embedded space from the eigenvectors corresponding to the k largest eigenvalues.
3) Grouping: Apply k-means to reduced $n \times k$ space to produce k clusters.

VI. Decoding and Interpretation

After extracting the patterns and modeling the data, the model is ready to be used to *decode* or *interpret* new observation sequences to answer some questions. For instance in the domain of biological sequence analysis, there is a need to know whether or not a protein shares similar properties with a number of other proteins. This is equivalent to asking if that protein contains the same patterns as the proteins belonging to the same family that is summarized in the model. Another question, in the domain of image segmentation, could be finding the "best" segmentation of scanned images of printed bilingual dictionaries [47] against a built model.

These questions are equivalent to finding the best state sequence in the model. This problem is referred to in HMMs [67] as the *decoding* problem. This problem is difficult to solve since it has several possible ways of solving it. In fact, finding the best state sequence is equivalent to finding the optimal state sequence that will decode or interpret the new observation sequence. Therefore, there are several optimality criteria. One possible solution would be to choose individually the most likely states q_t. Let the probability of being in state s_i at time t, given the observation sequence O, and the model λ, be defined as follows:

$$\gamma_t(i) = P(q_t = s_i | \lambda, O) \tag{37}$$

The partial observation sequence $o_1 o_2 \cdots o_t$ accounts for $\alpha_t(i)$ from the forward-backward algorithm, see Section 3.2.2 in Chapter 3. On the otherhand, the remainder of the observation sequence, $o_{t+1} o_{t+2} \cdots o_T$, accounts for $\beta_t(i)$ given state s_i at time t. Therefore, Equation 37 can be in terms of the forward-backward variables as follows:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{38}$$

Note that:

$$\sum_{i=1}^{N} \gamma_t(i) = 1 \tag{39}$$

Now the most likely state q_t at time t can be solved using: $\gamma_t(i)$ as follows:

$$q_t = arg \max_{1 \leq i \leq N} [\gamma_t(i)], \quad 1 \leq t \leq T. \tag{40}$$

Although choosing the most likely state for each time t appears to maximize the states that will explain the observation sequence O, this could result in a problem since it looks at the most likely state at each time t and ignores the probability of occurrence of sequences of the states. For instance, if a model has some transitions with zero probability between some states, then the optimal sate sequence could be an invalid, since the transition is not possible. To solve this problem, the optimization should be on the state sequence or path. This is equivalent to maximizing $P(Q, O \mid \lambda)$. The most widely used method to solve for this optimization problem is the Viterbi algorithm [38], [78], a technique based on dynamic programming.

For VOGUE, we can use the same concept to answer the question of interpretation. However, because of VOGUE's unique structure and needs, we modified the Viterbi algorithm to handle the notion of duration in the states. This is very important since VOGUE's gap states have the notion of duration. This changes the search for the optimal path to traverse the model's states as opposed to a regular HMM. We call this new proposed algorithm as, Variable-Gap Viterbi (VG-Viterbi).

The remainder of this chapter is organized as follows: first we give a description of the Viterbi algorithm since it is the basis for our VG-Viterbi. Then, we describe our proposed method, VG-Viterbi.

A. Viterbi Algorithm

Finding the best sequence of states q^* for the observation sequence $O = \{o_1 o_2 \cdots o_T\}$ given the model λ, in the Viterbi algorithm [38], [78], is equivalent to solving:

$$q* = arg \max_q P(q|\lambda, O) \tag{41}$$

where $q* = \{q_1^*, q_2^*, \cdots, q_T^*\}$. Now we need to define the highest probability along a single path, at time t, which accounts for the first observations in O and ends in state s_i by the quantity:

$$\delta_t(i) = \max_{q_1 q_2 \cdots q_{t-1}} P(q_1 q_2 \cdots q_t = i, q_{t+1} \neq i, o_1 o_2 \cdots o_t | \lambda) \tag{42}$$

By induction:

$$\delta_{t+1}(i) = [\max_i \delta_t(i) a_{ij}] \cdot b_j(o_{t+1}) \tag{43}$$

To retrieve the best state sequence q^*, the arguments which maximize Equation 43 need to be accounted for each t and j. This can be done by using:

$$\psi_t(j) = arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] \tag{44}$$

The procedure for finding the optimal path (sequence of states) that describes the observation sequence O, [78] is as follows:

1) Initialization:

$$\delta_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N \tag{45}$$

$$\psi_t(i) = 0. \tag{46}$$

2) Recursion:

$$\delta_t(i) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(o_t), \quad 2 \leq t \leq T, \quad 1 \leq j \leq N \tag{47}$$

$$\psi_t(j) = arg \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}], \quad 2 \le t \le T, \quad 1 \le j \le N \tag{48}$$

3) Termination:

$$p^* = \max_{1 \le i \le N} [\delta_T(i)] \tag{49}$$

$$q^* = arg \max_{1 \le i \le N} [\delta_T(i)]. \tag{50}$$

4) Path (state sequence) backtracking:

$$q_t^* = \psi_{t+1}(q^{*t+1}), \quad t = T-1, T-2, \cdots, 1. \tag{51}$$

B. VG-Viterbi: Variable Gap Viterbi

Once VOGUE is built to model a data set, and given a new sequence of observations $O = o_1 o_2 \cdots o_T$, there is a need to know whether this sequence belongs to the same class/family of the training data sequences. In other words, there is a need to interpret the new sequence based on the built model λ. This problem is equivalent to finding the best sequence of states from the model λ and gap state duration length that will describe the new sequence in an optimal and meaningful way. That is finding a sequence of states $q* = \{q_1^*, q_2^*, \cdots, q_T^*\}$ from the model λ such that:

$$q* = arg \max_q P(q|\lambda, O) \tag{52}$$

This is equivalent to finding the most probable path to be traversed in λ that would produce O. The algorithm that is mostly used to solve this problem is the Viterbi algorithm [38], [78]. Due to the unique structure of VOGUE, where gap states have a notion of duration, we adjusted the Viterbi algorithm accordingly. We will call this method Variable-Gap Viterbi (VG-Viterbi). We start from the basic formula in Equation52 that is used in Viterbi Algorithm and adjust as follows:

a) Definition: Let $\delta_t(j)$ be the highest probability path that produces the subsequence $O_t = o_1 o_2 \cdots o_t$ and terminates in state j at time t:

$$\delta_t(j) = \max_{q_1 \cdots q_{t-1}} P(o_1, \cdots, o_t, q_1, \cdots, q_{t-1}, q_t = j) \tag{53}$$

Equation (53) is equivalent to:

$$\begin{aligned}\delta_t(j) = \max_{\substack{r, s_1, \cdots, s_{r-1} \\ g_1, \cdots, g_r}} & P(o_1, \cdots, o_t, q_1 = \cdots = q_{g_1} = s_1, \\ & q_{g_1+1} = \cdots = q_{g_1+g_2} = s_2, \cdots, \\ & q_{1+\sum_{h=1}^{r-1} g_h} = \cdots = q_{\sum_{h=1}^{r} g_h} = s_r = j \mid \lambda)\end{aligned} \tag{54}$$

where $g_h \in \{1, \cdots, maxgap\}$ is the duration of staying in a state. The maximum of the probability in Equation(54) is taken such that $\sum_{h=1}^{r} g_h = \min(t, maxgap)$, $s_h \in \{1, \cdots, N\}$, $h = 1, \cdots, r$, $s_l \neq s_{l+1}$, $l = 1, \cdots, r-1$ and $1 \le r \le \min(t, maxgap)$.

If we omit the qs from Equation(54):

$$\begin{aligned}\delta_t(j) = \max_{\substack{g_r, i \neq j, s_1, \cdots, s_{r-2} \\ g_1, \cdots, g_{r-1}}} & P(o_1, \cdots, o_t, s_1, \cdots, s_{r-1} = i, \\ & s_r = j, g_1, \cdots, g_r \mid \lambda)\end{aligned} \tag{55}$$

Applying Bayes rule we obtain:

$$\begin{aligned}\delta_t(j) = & \max_{\substack{g_r, i \neq j, s_1, \cdots, s_{r-2} \\ g_1, \cdots, g_{r-1}}} P(o_1, \cdots, o_t, s_1, \cdots, s_{r-2}, s_r = \\ & j, g_1, \cdots, g_r \mid s_{r-1} = i, \lambda) \cdot P(s_{r-1} = i \mid \lambda) \\ = & \max_{\substack{g_r, i \neq j, s_1, \cdots, s_{r-2} \\ g_1, \cdots, g_{r-1}}} P(o_1, \cdots, o_{t-g_r}, s_1, \cdots, s_{r-2}, g_1, \cdots, g_r \mid \\ & s_{r-1=i}, o_{t-g_r+1}, \cdots, o_t, s_r = j, g_r, \lambda) \\ & \cdot P(o_{t-g_r+1}, \cdots, o_t, s_r = j, g_r \mid s_{r-1} = i, \lambda) \\ & \cdot P(s_{r-1} = i \mid \lambda)\end{aligned} \tag{56}$$

Applying the “Markovian” assumption, the current state depends only on the previous state, therefore we obtain :

$$\begin{aligned}\delta_t(j) \quad = \max_{\substack{g_r, i \neq j, s_1, \cdots, s_{r-2} \\ g_1, \cdots, g_{r-1}}} & P(o_1, \cdots, o_{t-g_r}, s_1, \cdots, s_{r-2}, \\ & g_1, \cdots, g_{r-1} \mid s_{r-1} = i, \lambda) \\ & \cdot P(o_{t-g_r+1}, \cdots, o_t, s_r = j, g_r \mid s_{r-1} = i, \lambda) \\ & \cdot P(s_{r-1} = i \mid \lambda)\end{aligned} \tag{57}$$

By combining the first and last terms, and using Bayes' rule on the second term in Equation(57), we obtain:

$$\begin{aligned}\delta_t(j) \quad = \max_{\substack{g_r, i \neq j, s_1, \cdots, s_{r-2} \\ g_1, \cdots, g_{r-1}}} & P(o_1, \cdots, o_{t-g_r}, s_1, \cdots, s_{r-2}, \\ & s_{r-1} = i, g_1, \cdots, g_{r-1} \mid \lambda) \\ & \cdot P(o_{t-g_r+1}, \cdots, o_t, g_r \mid s_r = j, s_{r-1} = i, \lambda) \\ & \cdot P(s_r = j \mid s_{r-1} = i, \lambda))\end{aligned} \tag{58}$$

Let's assume that the duration distribution of a state is independent of the observations of that state. Since $\sum_{h=1}^{r} g_h = min(t, maxgap)$ then $g_r < min(t, maxgap)$. Moreover, since the observations are independent from each other given their states, we obtain:

$$\begin{aligned}\delta_t(j) \quad = & \max_{\substack{g_r < min(t, maxgap) \\ i \neq j}} \delta_{t-g_r}(i) \cdot P(s_r = j \mid s_{r-1} = i, \lambda) \\ & \cdot \quad P(g_r \mid s_r = j, \lambda) \quad \cdot \quad [\prod_{s=t-g_r+1}^{t} P(o_s \mid s_r = j, \lambda)] \\ & = \max_{g_r, i \neq j} \delta_{t-g_r}(i) \cdot \beta_{ij} \cdot \rho_{jg_r} \cdot [\prod_{s=t-g_r+1}^{t} b_{js}]\end{aligned} \tag{59}$$

For simplicity we denote g_r by g, then we get the following recursive relationship:

$$\delta_t(j) = \max_{\substack{g < min(t, maxgap) \\ i \neq j}} \delta_{t-g}(i) \cdot \beta_{ij} \cdot \rho_{jg} \cdot [\prod_{s=t-g+1}^{t} b_{js}] \tag{60}$$

where:

$$\beta_{ij} = \begin{cases} a_{ij}, & \text{if } g < min(t, maxgap) \\ a_{Nj}, & \text{if } g = min(t, maxgap) \text{ or } t = 1 \end{cases} \tag{61}$$

The initialization for VG-Viterbi is:

$$\delta_0(j) = \begin{cases} 1, & \text{if } j = N \\ 0, & \text{otherwise} \end{cases} \tag{62}$$

$$\psi_0(j) = \begin{cases} j, & \text{if } j = N \\ 0, & \text{otherwise} \end{cases} \tag{63}$$

Therefore VG-Viterbi algorithm is defined as:

1) Initialization:

$$\delta_0(j) = \begin{cases} 1, & \text{if } j = N \\ 0, & \text{otherwise} \end{cases} \tag{64}$$

$$\psi_0(j) = \begin{cases} j, & \text{if } j = N \\ 0, & \text{otherwise} \end{cases} \tag{65}$$

2) Recursion:

$$\delta_t(j) = \max_{\substack{g < min(t, maxgap) \\ i \neq j}} \delta_{t-g}(i) \cdot \beta_{ij} \cdot \rho_{jg} \cdot [\prod_{s=t-g+1}^{t} b_{js}] \tag{66}$$

where:

$$\beta_{ij} = \begin{cases} a_{ij}, & \text{if } g < min(t, maxgap) \\ a_{Nj}, & \text{if } g = min(t, maxgap) \text{ or } t = 1 \end{cases} \tag{67}$$

C. VG-Viterbi: optimization

Since VG-Viterbi is based on the Viterbi algorithm [78], it inherits its advantages and drawbacks. One of the drawbacks of the Viterbi algorithms is that it exhaustively searches the state space of the HMM to find the best (*optimal*) path of state sequences to describe the observation sequence O. If N is the number of states of the model λ, and T is the length of O, then the Viterbi algorithm's time complexity is $O(N^2 \times T)$ [38], [78]. This is obviously expensive when the number of states N is large and the observation sequence is very long. In fact in some areas like in biological sequence analysis [26], where the length of a sequence varies between 500 and 1000 elements, and when the HMM is about 900 states the estimated time complexity of the Viterbi algorithm is on the order of 8×10^8. The time complexity of the VG-Viterbi algorithm is comparable to that of the Viterbi algorithm. In fact, let N be the number of states in VOGUE, and the length of the observation sequence O be T. In the case of an exhaustive search of the model λ, for every element O_t in O $(0 \le t \le T)$, we need to check for the highest probability of O_t to be emitted by a state q_i where $1 \le i \le N$. This is done by taking into consideration the state q_j that has the highest probability of emitting O_{t-1}. Moreover, some of the states are "gap" states, and they have a notion of duration up to $maxgap$ times. Hence, we need to explore the number of O's elements that will be emitted from the same "gap" state, i.e., "staying" in the same state for up to $maxgap$. Therefore, the estimated time is $O(N \times N \times maxgap \times T)$. In order to reduce the time complexity we need to reduce either N, $maxgap$ or T. Since we cannot reduce T, which is the length of the observation sequence, we have to explore reducing either $maxgap$ or N.

VOGUE is not a fully connected graph, since all the states are not connected to all the states. In fact, the only possible transitions, whose probability is nonzero are the following transitions:

- Transitions from the *Universal Gap* state to the *first state symbols* $q_j \in Q_f$ and to the *intermediate gap states* $q_j \in Q_i$. Therefore the number of allowed transitions (nonzero transitions) from the *Universal Gap state* is $\mid Q_f \mid + \mid Q_i \mid$.
- Transitions from the *first state symbols* $q_i \in Q_f$ to a *second state symbol* $q_j \in Q_s$ (modeling a gap of $g = 0$) or to an *intermediate gap state* $q_j \in Q_i$. Therefore the number of allowed transitions (nonzero transitions) from the *first state symbols* is $\mid Q_s \mid + \mid Q_i \mid$.
- Transitions from the *intermediate gap state* $q_i \in Q_i$ to only a *second state symbol* $q_j \in Q_s$. Therefore the number of allowed transitions (nonzero transitions) from the *intermediate gap state* is $\mid Q_s \mid$.
- Transitions from the *second symbol state* $q_i \in Q_s$ to a *first state symbol* $q_j \in Q_f$ (modeling a gap of $g = 0$) or to the *Universal Gap state* $q_j \in Q_u$. Therefore the number of allowed transitions (nonzero transitions) from the *intermediate gap state* is $\mid Q_f \mid + \mid Q_u \mid$.

Therefore the transition matrix A is a sparse matrix as shown in Equation 68. For every observation O_t in O we don't need to do an exhaustive search of all the states in VOGUE to find the state that will emit O_t such that $\delta_{t-g}(i) \cdot \beta_{ij} \cdot \rho_{jg} \cdot [\prod_{s=t-g+1}^{t} b_{js}]$ is maximal, as described in Equation 60. In fact, not all the transitions from state q_t to the states $q_j \in Q$ are nonzero.

Considering some of the transitions in VOGUE are non-existent ($a_{ij} = 0$), we propose the following ***Recursion*** step of VG-Viterbi:

$$\delta_t(j) = \begin{cases} \max\limits_{\substack{g<min(t,maxgap)\\ Q_f<i\le N-1}} \omega_t(g,i), & \text{if } 1 \le j \le Q_f \\ \max\limits_{\substack{g<min(t,maxgap)\\ Q_f+Q_i\le i\le N-1}} \omega_t(g,i), & \text{if } Q_f < j \le Q_f + Q_i \\ \max\limits_{\substack{g<min(t,maxgap)\\ i\in\{1,\cdots,Q_f\}\cup\{N\}}} \omega_t(g,i), & \text{if } Q_f + Q_i < j \le N-1 \\ \max\limits_{\substack{g<min(t,maxgap)\\ 1\le i\le Q_f}} \omega_t(g,i), & \text{if } j = N \end{cases} \tag{69}$$

where:

$$\omega_t(g,i) = \delta_{t-g}(i) \cdot \beta_{ij} \cdot \rho_{jg} \cdot [\prod_{s=t-g+1}^{t} b_{js}] \tag{70}$$

and

$$\beta_{ij} = \begin{cases} a_{ij}, & \text{if } g < min(t,maxgap) \\ a_{Nj}, & \text{if } g = min(t,maxgap) \text{ or } t = 1 \end{cases} \tag{71}$$

In this case, we don't iterate through all the values of $i \in \{1, \cdots, N\}$ since some of the values of a_{ij} and hence β_{ij} are zero depending on the state q_j (i.e. the value of j in $\{1, \cdots, N\}$). For example, if $1 \le j \le Q_f$, $\beta_{ij} \ne 0$, only when $Q_f < i \le N-1$ as shown in the transition matrix A in Equation(68).

Therefore, the time complexity of the optimized VG-Viterbi is:

$$\tau = O(\{[Q_f \times (Q_i + Q_s)] + [Q_f \times Q_s] + [Q_s \times (Q_f + 1)]\} \times T \times maxgap) \tag{72}$$

which becomes:

$$\begin{aligned} \tau \quad &= O(\{[Q_i \times (Q_f + Q_s)] + \\ &2Q_s \times [Q_f + 1]\} \times T \times maxgap) \\ &= O(\{Q_i \times (Q_f + Q_s)\} \times T \times maxgap) + \\ &2O(\{Q_s \times (Q_f + 1)\} \times T \times maxgap) \end{aligned} \tag{73}$$

Q_f and Q_s never exceed M the number of observations in VOGUE since they are the number of "distinct" first and second symbols in the mined sequences by VGS, respectively. In fact there could be at most M different symbols identified by VGS. Since M is always a fixed number as opposed to the number of sequences retained by VGS, reflected by Q_i, when N is large Q_i large. Therefore, $O(Q_i \times (Q_f + Q_s) \times T \times maxgap) \le O(Q_i \times 2M \times T \times maxgap)$, and $O(Q_i \times 2M \times T \times maxgap) \simeq 2MO(Q_i \times T \times maxgap) \simeq O(Q_i \times T \times maxgap)$. Likewise, $O(Q_s \times (Q_f + 1) \times T \times maxgap) \le O(M(M+1) \times T \times maxgap)$ and $O(M(M + 1) \times T \times maxgap) \simeq O(T \times maxgap)$.

Therefore, $\tau \preceq O((Q_i + 1) \times T \times maxgap) \ll O(N^2 \times T \times maxgap)$, since $Q_i < N$.

VII. Experimental Results and Analysis

Several real world applications, such as in bioinformatics, web accesses, finance, and text mining, encompass sequential and temporal data with long range dependencies. The fact that VOGUE has been designed to capture and model such long range dependency patterns, makes it a very good tool to model some of those applications. In [2], we developed a toolkit to facilitate the sharing and long term use of different types of geological data sets across disciplines. Geoscientists are confronted with an enormous quantity of diverse data types. These data sets include published data, geological and topological maps, satellite imagery, structural, seismic, geophysical, petrologic and geochronologic data. The time when the data are added or accessed by users is recorded. While each piece of data originates at a specific user, each user is allowed to add new annotations to the data as they wish. Finding where the data is and what type of data a user should access next is a challenging problem. We use VOGUE as the core of this toolkit to model user access patterns

$$A = \begin{array}{cc}
 & \begin{array}{cccccccccc} & Q_f & & & Q_i & & & Q_s & & Q_u \end{array} \\
\begin{array}{c} \\ Q_f \\ \\ \\ Q_i \\ \\ \\ Q_s \\ \\ Q_u \end{array} &
\begin{pmatrix}
0 & \dots & 0 & a_{1,j+1} & \dots & a_{1,k} & a_{1,k+1} & \dots & a_{1,N-1} & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & \dots & 0 & a_{j,j+1} & \dots & a_{j,k} & a_{j,k+1} & \dots & a_{j,N-1} & 0 \\
0 & \dots & 0 & 0 & \dots & 0 & a_{j+1,k+1} & \dots & a_{j+1,N-1} & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & \dots & 0 & 0 & \dots & 0 & a_{k,k+1} & \dots & a_{k,N-1} & 0 \\
a_{k+1,1} & \dots & a_{k+1,j} & 0 & \dots & 0 & 0 & \dots & 0 & a_{k+1,N} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
a_{N-1,1} & \dots & a_{N-1,j} & 0 & \dots & 0 & 0 & \dots & 0 & a_{N-1,N} \\
a_{N,1} & \dots & a_{N,j} & 0 & \dots & 0 & 0 & \dots & 0 & 0
\end{pmatrix}
\end{array} \tag{68}$$

Fig. 5. VG Viterbi Matrix where $j = Q_f$ and $k = Q_f + Q_i$.

of the data in the database. Moreover, it allows a new user to visualize the most common patterns of use or the latest common patterns of use based on the data set used to construct the VOGUE state machine. When a user accesses and adds new data to the database, he/she starts a new project. A specific project may have multiple patterns of use at the same granularity corresponding to different research activities or problems being investigated. We built different VOGUE state machines to accommodate the multiple views corresponding to different interpretations of the data.

Here, we describe several experiments that we conducted to compare the performance of VOGUE, C-VOGUE and K-VOGUE to those of two popular techniques in the domain of biological sequence analysis: HMMER and all-kth-order HMM. We used two data sets, a real data set from the PROSITE database, a sequential data base of families of proteins, and the SCOP data set, a manually derived comprehensive hierarchical classification of known proteins structures that has secondary structure knowledge embedded in the data set.

A. Protein modeling and clustering using VOGUE

The completion of the whole genome sequencing of various organisms facilitates the detection of many kinds of interesting patterns in DNA and protein sequences. It is known that the genomes of most plants and animals contain large quantities of repetitive DNA fragments or, in the case of proteins, Amino Acid fragments. For instance, it is estimated that one third of the human genome is composed of families of repeating sequences [31], [32], [93]. The amino acids are thus far from being pieces of random sequences, and a substantial amount of currently unknown information can be extracted from the sequences in the form of patterns. The abundance and variety of periodic patterns in genome sequences drove a lot of studies on genome sequence analysis and mining. In fact, periodic patterns of different lengths and types are found at both geneomic and proteomic levels. The short three base pair (*bp*) periodicity in protein coding DNA [37], and the medium-length repetitive motifs found in some proteins [20], to the mosaic of very long DNA segments in the genome of warm-blooded vertebrates [15], are some of these patterns. It is very important to identify some of these patterns due to their biological significance. For instance, some repeats have been shown to affect bacterial virulence to human being [77]. On the other hand, the excessive expansions of some Variable Number of Tandem Repeats ($VNTRs$) are the suspected cause of some nervous system diseases [68]. Therefore, there is a growing need for efficient algorithms to extract periodic patterns from long sequences.

In recent years, a large amount of work in biological sequence analysis has focused on methods for finding homologous proteins [26]. Given a database of protein sequences, the goal is to build a statistical model so that we can determine whether a query protein belongs to a given family or not. HMMER [29], a profile HMM, is one of the state-of-the-art approaches to this problem that depends heavily on a good multiple sequence alignment. It models gaps, provided that they exist in the alignment of *all* the training sequences. However, if a family of sequences has several overlapping motifs, which may occur in different sequences, these sequences will not be aligned correctly, and HMMER will not perform well. Here, we analyze the performance of VOGUE compared to HMMER and higher-order HMMs with various orders $k \in [1, 10]$.

Computationally, protein sequences are treated as long strings of characters with a finite alphabet of 20 amino acids. Namely, A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, and Y. There are many patterns depending on the issues considered, for example the number of periods of the patterns, the maximality of the patterns, whether errors (insertions, deletions and substitutions) are allowed and palindromic reverses. In [50], the authors provide a survey on studies to extract patterns taking into consideration one of the issues mentioned earlier.

In this work, we are particularly interested in extracting patterns that identify a family of proteins. The first method focuses on extracting those patterns from the sequences formed of the 20 amino acids and not allowing any substitutions between them. The second method, extracts patterns while allowing for substitutions between amino acids that have similar structure and functionality (Hydrophobicity, Charge, Polarity, etc).

We apply VOGUE, C-VOGUE, and K-VOGUE to a real world problem, namely, finding homologous proteins. Given a database of protein sequences, the goal is to build a statistical model so that we can determine whether a query protein belongs to a given family or not. Statistical models for proteins, such as profiles, position-specific scoring matrices, and hidden Markov models [26] have been developed to find homologs. However, in most biological sequences interesting patterns are periodic with gap requirements. Therefore a method like VOGUE that specifically takes these kind of patterns into consideration can be very effective. We show experimentally that VOGUE's modeling power is superior to higher-order HMMs while reducing the latter's state-space complexity, and improving their prediction. VOGUE also outperforms HMMER [29], a HMM model especially designed for protein sequences that takes into consideration insertions, deletions and substitutions between "omo*similar*" amino acids. We will give a an overview of HMMER in Section VII-A.1. Then, we will describe the scoring and evaluation measure we use for evaluating the performance of the methods used. Afterwards, we describe the data sets that we use for our experimentation. Finally, we provide the performance results of VOGUE, C-VOGUE and K-VOGUE vs HMMER and higher-order HMM.

1) HMMER: HMMER [29] is a HMM model especially designed for protein sequences that takes into consideration insertions, deletions and substitutions between "omo*similar*" amino acids. It is called a "Profile" HMM, a well suited HMM for multiple alignments of sequences. We will to first describe what a multiple alignment is, and then describe a "Profile" HMM.

Multiple Alignment: the problem of multiple alignment is described as follows: Given a set of sequences, produce a multiple alignment which corresponds as well as possible, to the biological relationships between the corresponding bio-molecules [26]. Two amino acids should be aligned (on top of each other) in the following conditions:

- if they are homologous (evolved from the same residue in a common ancestor).
- if they are structurally equivalent.

To identify whether an alignment is good, a *fitness* function is used where the biological relationships are taken into considerations. For example, assuming the following three sequences:

$$\begin{array}{cccccccccc} I & N & D & U & S & T & R & Y & & \\ I & N & T & E & R & E & S & T & I & N & G \\ I & M & P & O & R & T & A & N & T & \end{array}$$

One alignment could be :

$$\left|\begin{array}{c|c|c|c|c|c|c|c|c|c|c} I & N & - & D & U & - & S & T & R & Y & - \\ I & N & T & E & R & E & S & T & I & N & G \\ I & M & - & P & O & R & - & T & A & N & T \end{array}\right|$$

But the following is not a good alignment based on the biological characteristics of the amino acids:

$$\left|\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c} I & N & - & D & U & - & - & S & T & R & Y & - \\ I & N & T & E & R & E & - & S & T & I & N & G \\ I & M & - & P & O & R & - & - & T & A & N & T \end{array}\right|$$

For a more detailed description of the multiple alignment process and the different available methods refer to [61].

Profile HMM: One of the general features of protein family multiple alignment is that "gaps" tend to line up with each other, leaving solid blocks of either exact matches or allowed substitutions between the amino acids. These positions are considered to be the "ungapped" states of the HMM. The emission probability is based on a *position specific score matrix* (PSSM). More details can be found in [26]. Thus, the HMM is built with a repetitive structure of states but different probabilities in each one in a left-to-right manner. The PSSM is a HMM with a series of identical states, called "match states" (M_j), separated by transitions of probability 1. Although PSSM captures some conservation information, it is does not represent all the information in a multiple alignment of a protein family. Therefore, "Insertion" states (I_j) are introduced in the HMM, where each of the I_j states is used to match insertion after the element emitted by the matching state M_j. "Deletion" states (D_j) are also added to act as silent states that do not emit any symbol. Therefore, it is possible to use them to "jump" from any "match" state to another one without emitting any symbol in between. HMMER is a software that is based on building profile HMMs to model protein families.

2) Evaluation and Scoring: We built three models for each family, namely VOGUE, HMMER, and k-th order HMMs, using the training set of that family. We score the test sequences against the model for each of the nine families, and after sorting the scores in decreasing order, we use a threshold on the scores to assign a sequence to a given family.

For evaluation of the classifiers, we use Receiver Operating Characteristic (ROC) curves [33], that represent the relationship between the false positive rate and true positive rate across the full spectrum of threshold values. Further, we plot the Area Under the Curve (AUC), to evaluate the goodness of the classifiers. The AUC is calculated using the following equation [33]:

$$\text{AUC} = \frac{1}{pn} \sum_{i=1}^{p} \sum_{j=1}^{n} \varphi(R_i, R_j). \tag{74}$$

Here $N_{test} = n + p$ is the number of testing sequences, p is the number of sequences from a given class and n is the number of sequences that don't belong to the class. These sequences are ranked based on their score from 1 to N_{test}, assigning 1 to the testing sequence with the highest score and N_{test} to the one with the lowest score. R_i, $i = 1 \cdots p$ represent the rankings of the p sequences and R_j, $j = 1 \cdots n$ represent the rankings of the n sequences and $\varphi(R_i, R_j)$ is defined as:

$$\varphi(R_i, R_j) = \begin{cases} 1 & \text{if } R_i < R_j \\ 0 & \text{otherwise} \end{cases} \tag{75}$$

Note that AUC for each class is calculated separately, by treating each class as p, and the remaining as n. We score the testing sequences by computing the log-odds score, (i.e., the ratio of the probability of the sequence using a given model, and the probability of the sequence using a **Null** model, given as follows:

$$\text{Log-Odds}(seq) = \log_2 \left(\frac{P(seq|Model)}{P(seq|\mathbf{Null})} \right). \tag{76}$$

$P(seq/Model)$ is computed using the Viterbi algorithm that computes the most probable path through the model) as Viterbi is the default method used for scoring in HMMER. The **Null** model is a simple one state HMM that emits the observations (the amino acids) with equal probability ($1/|\Sigma|$). Since we have 20 amino acids the emission probability for each symbol is $1/20$. The log-odds ratio measures whether the sequence is a better match to the given model (if the score is positive) or to the null hypothesis (if the score is negative). Thus, the higher the score the better the model.

3) Datasets: We used in our experiments two different data sets: a set of 9 families downloaded from the PROSITE (`http://www.expasy.org/prosite`) database of protein family and domains, and SCOP [19] data set, a manually derived comprehensive hierarchical classification of known protein structures, that are organized according to their evolutionary and structural relationships.

The PROSITE families that we used are $PDOC00662$, $PDOC00670$, $PDOC00561$, $PDOC00064$, $PDOC00154$, $PDOC00224$, $PDOC00271$, $PDOC00397$, $PDOC00443$. We will refer to these families as F_1, F_2, $\cdots$, F_9, respectively. The number of sequences in each family is, respectively: $N^1 = 45$, $N^2 = 225$, $N^3 = 85$, $N^4 = 56$, $N^5 = 119$, $N^6 = 99$, $N^7 = 150$, $N^8 = 21$, $N^9 = 29$. The families consists of sequences of lengths ranging from 597 to 1043 characters, taken from the alphabet of the 20 amino acids: $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$. Each family is characterized by a well-defined motif. Family F_1, for example, shares the consensus motif $[G] - [IVT] - [LVAC] - [LVAC] - [IVT] - [D] - [DE] - [FL] - [DNST]$, which has 9 components. Each component can contain any of the symbols within the square brackets. For example, for the second component, namely $[IVT]$, either I, V or T may be present in the sequences. We treat each PROSITE family as a separate class. We divided the data set of each family F_i into two subsets: the training data N^i_{train} consists of 90% of the data, while the testing data N^i_{test} contains the remaining

TABLE XVI

RUN TIMES

VOGUE	HMMER	$k=1$	$k=2$	$k=4$	$k=10$
$4.6s$	$34.42s$	$2s$	$5.29s$	$6.40s$	$11.46s$

10%. For example, $N^1_{train} = 40$ and $N^1_{test} = 5$. There are a total of 103 test sequences across all families.

The Scop data set is divided into four hierarchical levels: Class, Fold, Superfamily and Family. For SCOP 1.61 (from 2002), the 44327 protein domains were classified into 701 folds, resulting in an average of 64 domains per fold. The number of domains per fold varies in SCOP, where some of the folds, such as TIM barrels, are highly populated, while some of the folds, such as the $HSP40/DnaJ$ peptide-binding fold that only contain one protein, contain a few examples. Therefore, the SCOP is an imbalanced data set. This imbalanced proportion of examples in each fold contributes to the poor performance of classical machine learning techniques such as support vector machines and neural networks [23]. When learning from such data sets, existing machine learning approaches tend to produce a strong discriminatory classifier or ”high accuracy” with very low sensitivity or completeness. We used 10 superfamilies from the SCOP data set (`ftp://ftp.rcsb.org/pub/pdb/derived_data/pdb_seqres.txt`) namely, family 49417, 46458, 46626, 46689, 46997, 47095, 47113, 48508, 69118, and 81296. We will refer to them as SF_1, SF_2, SF_3, SF_4, SF_4, SF_5, SF_6, SF_7, SF_8, SF_9 and SF_{10} respectively. Each family has 10 sequences. We divided each family data set into 90% (9 sequences for each family)for training and 10% for testing (1 for each family to a total of 10 sequences).

B. Performance of VOGUE vs HMMER vs k-th Order HMMs on PROSITE data

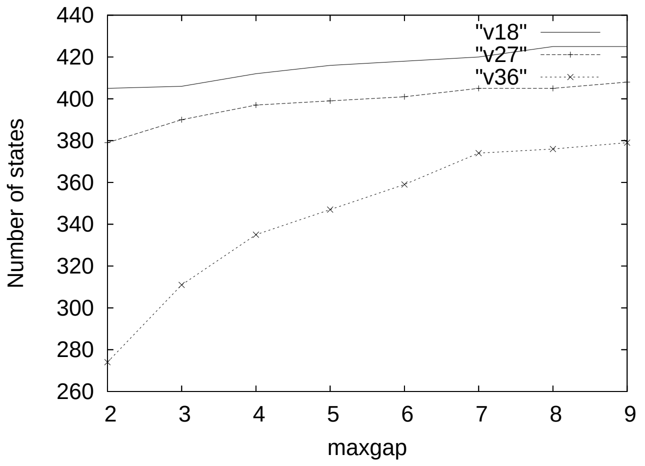


Fig. 6. VOGUE: Number of States for Different Parameters.

We built VOGUE state machines with different values of $minsup$ corresponding to 50%, 75% and 100% of the number of instances in the training data, and $maxgap$ (10, 15, 20, 25, 30) but with the constant $k = 2$ for the length of the mined sequences in VGS. We then choose the best set of parameters and fix them for the remaining experiments. To model the data using HMMER, we first need to align the training sequences using CLUSTAL-W (`http://www.ebi.ac.uk/clustalw`). We then build a profile HMM using the multiple sequence alignment and compute the scores for each test sequence using HMMER, which directly reports the log-odds scores with respect to the **Null** model mentioned above. We also built several k-th order HMMs for various values of k using an open-source HMM software (`http://www.cfar.umd.edu/~kanungo/software`). We tried different values for the number of states ranging from the size of the protein alphabet (20) to roughly the size of VOGUE (500) and HMMER (900). A k-th order HMM is built by replacing each consecutive subsequence of size k with a unique symbol. These different unique symbols across the training and testing sets were used as observation symbols. Then we model the resulting sequence with a regular 1st order HMM.

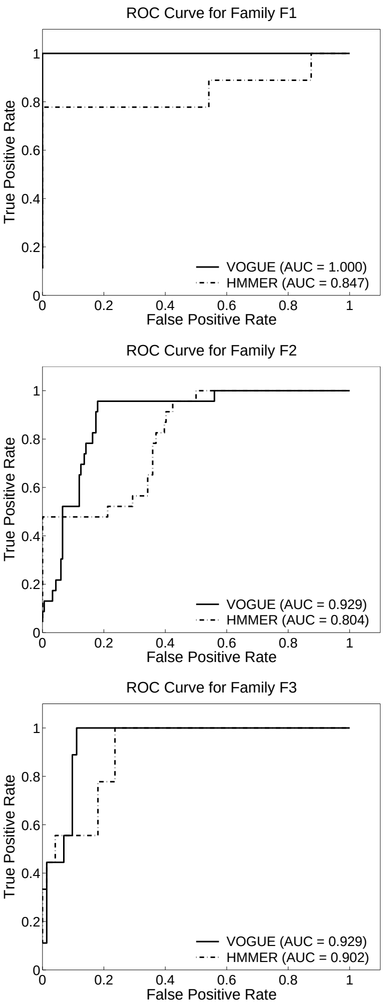


Fig. 7. ROC Curve of VOGUE and HMMER for the families F_1, F_2 and F_3.

Score Comparison: We first compare VOGUE with k-order HMMs and HMMER. Table XV shows the comparison on the 5 test sequences for family F_1 when scored against the model for F_1. For VOGUE we used $minsup = 27$ (75%) and $maxgap = 20$. For k-th order HMMs we tried several values of the order k (shown as $k = 1$, $k = 2$, $k = 4$, $k = 8$ and $k = 10$) in the table with 20 states for each k-th order HMM. The number of observations M for the $k = 1$ case was set to 20 since it is the number of amino acids. $M = 394; 17835; 20216; 19249$ were the number of observations used for $k = 2; 4; 8; 10$, respectively. These values were obtained from a count of the different new symbols used for each value of k.

The best score for each sequence is highlighted in bold. Looking at the scores in Table XV, we find that in general k-th order HMMs were not able to model the training sequences well. All their scores are large negative values. HMMER did fairly well, which is not surprising, since it is specialized to handle protein

TABLE XV
TEST SEQUENCE LOG-ODDS SCORES FOR VOGUE, HMMER AND k-TH ORDER HMMS

Seq	S_1	S_2	S_3	S_4	S_5
VOGUE	**7081**	**7877**	**2880**	**5763**	**5949**
HMMER	912.4	155	-345	9.8	-21.3
k-th order HMM $k = 1$ $M = 20$	-4×10^3	-3.4×10^3	-2.2×10^3	-4.7×10^3	-4.7×10^3
k-th order HMM $k = 2$ $M = 394$	-1.3×10^4	-1.3×10^4	-1×10^4	-1.5×10^4	-1.5×10^4
k-th order HMM $k = 4$ $M = 17835$	-2.3×10^4	-2.2×10^4	-1.8×10^4	-2.4×10^4	-2.4×10^4
k-th order HMM $k = 8$ $M = 20216$	-2×10^4	-1.9×10^4	-1.6×10^4	-2.2×10^4	-2.2×10^4
k-th order HMM $k = 10$ $M = 19249$	-2.6×10^4	-2.9×10^4	-2.3×10^4	-3.0×10^4	-3.1×10^4

sequences. Moreover, for all the 5 testing sequences VOGUE vastly outperforms HMMER. This is a remarkable result when we consider that VOGUE is completely automatic and does not have explicit domain knowledge embedded in the model, except what is recovered from relationship between symbols in the patterns via mining.

Time Comparison: In Table XVI, we show the execution time for building the three models for family F_1. The time for VOGUE includes the mining by VGS, and for HMMER, the alignment by CLUSTAL-W. In general, for VOGUE, the higher the minimum support, the lower the running time, and the higher the maximum gap, the higher the running time; the running time of VOGUE varied from 2.6s (for $minsup = 36, maxgap = 10$) to 4.6s ($minsup = 18, maxgap = 30$). We can see that VOGUE's execution time is in general much better than HMMER and is also better than higher-order HMMs (except for $k = 1$). Thus not only is VOGUE more accurate in modeling the input, it also executes faster.

Size Comparison: We also compared the state space complexity of the three methods. The number of states in HMMER was $N = 971$, while for higher-order HMMs it ranged from 500 to 900. VOGUE on the other hand was able to reduce the state space complexity by only modeling the mined sequences and not the full data set thus eliminating noise. Figure 6 shows the number of states in VOGUE for various $maxgap$ and $minsup$ values. We find that varying the parameters for VOGUE does not alter the state space complexity considerably. The biggest number of states, $N = 425$, is for $minsup = 18$ and $maxgap = 9$; and the smallest, $N = 274$, for $minsup = 36$ and $maxgap = 2$. This follows from the fact that the higher the $minsup$ the less the frequent sequences mined by VGS, and vice versa.

Full Comparison (ROC Curves and AUC): Figures 7, 8 and 9 present the ROC curves of the 9 families generated from all the testing sequences. Here we focus on comparing HMMER and VOGUE, since k-th order HMMs gave highly negative scores for all the testing sequences. The ROC curves represent the trade-off between coverage (TPR on the $yaxis$) and error rate (FPR on the $xaxis$) of a classifier. A good classifier will be located at the top left corner of the ROC graph, illustrating that this classifier has high coverage of true positives with few false positives. A trivial rejector will be at the bottom left corner of the ROC graph and a trivial acceptor will be at the top right corner of the graph. Each one of the graphs in Figures 7, 8, and 9 has two ROC curves for VOGUE and HMMER, respectively, for different threshold values. The total AUC for the two methods is given in the Figure legend. VOGUE was run with typical parameter values of $minsup = 75\%$ and $maxgap = 20$; there were some minor variations to account for characteristics of different families. The ROC curves of all the families show clearly that VOGUE improved the classification of the data over HMMER because the AUC of VOGUE is constantly higher than HMMER. In the case of family F_9 the AUC of both VOGUE and HMMER were comparable. In two cases, for families F_1 and F_6, the AUC was 1 for VOGUE showing that VOGUE was able to capture the patterns of those families perfectly. Moreover, in 6 out 9 families the AUC for VOGUE was higher than 0.9 as opposed to HMMER whose AUC was greater than 0.9 in only 3 out of 9 families. This again shows that VOGUE outperforms HMMER.

C. Performance of VOGUE vs C-VOGUE vs HMMER on PROSITE data

In this section, besides the models of VOGUE and HMMER from the previous section, we also built a C-VOGUE state machine for the PROSITE data set. We first run VGS on the training data set with different values of $minsup$ corresponding to 50%, 75% and 100% of the number of instances in the training data set, and $maxgap$ $(10, 15, 20, 25, 30)$ but still with the constant $k = 2$ as the length of the mined sequences by VGS. Then we prune the "artifacts" from the set of frequent sequences and we build the new model C-VOGUE. We then choose the best set of parameters, and fix them for the remaining of the experiments. We then compare C-VOGUE to VOGUE and to HMMER. The results of C-VOGUE as opposed to VOGUE and HMMER are shown in Figures 10, 11 and 12.

These figures show clearly that VOGUE's and C-VOGUE's ROC curves overlap for all 9 families, hence have the same

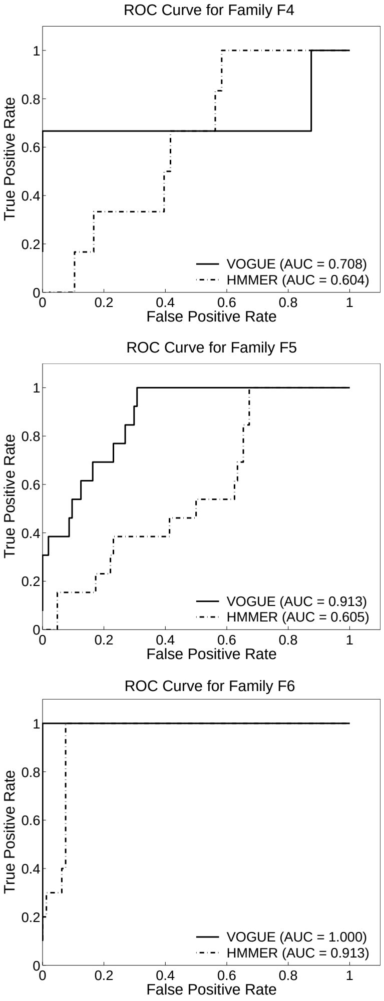


Fig. 8. ROC Curve of VOGUE and HMMER for the families F_4, F_5 and F_6.

TABLE XVII
THE NUMBER OF STATES N USING VOGUE VS C-VOGUE FOR THE 9 PROSITE FAMILIES

Family	VOGUE	C-VOGUE
F_1	421	305
F_2	350	252
F_3	421	301
F_4	423	309
F_5	375	268
F_6	408	290
F_7	298	222
F_8	421	318
F_9	420	319

AUC. Therefore, C-VOGUE also outperforms HMMER. This experiments reinforces the claim that C-VOGUE keeps a good coverage and an increased accuracy. Concerning the state space complexity, Table XVII shows the number states using VOGUE and C-VOGUE for the 9 families.

The number of states of the families models using C-VOGUE is clearly smaller by 27% than that of the families models using VOGUE. Therefore, these experiments show the benefit of pruning using C-VOGUE in reducing the state space complexity while preserving the good coverage and accuracy of VOGUE.

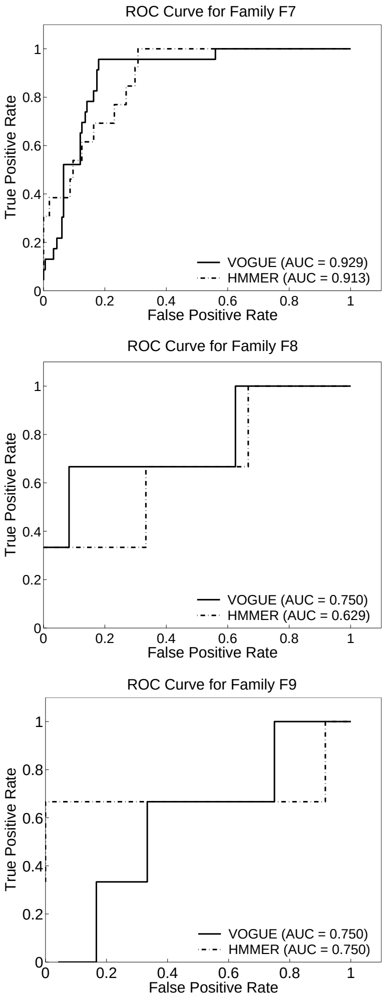


Fig. 9. ROC Curve of VOGUE and HMMER for the families F_7, F_8 and F_9.

TABLE XVIII
THE 9 CLUSTERS FOR THE AMINO ACIDS PROVIDED BY THE DOMAIN EXPERT

Cluster	Elements	Description
C_1	H, R, K	Positively charged
C_2	A, L, V, I, M	Aliphatic. M is the exception, but it is hydrophobic and can fit here
C_3	F, Y, W	Aromatic amino acids
C_4	D, E	Negatively charged
C_5	P	Aliphatic with a pseudo ring
C_6	S, T	With hydroxyl side chains
C_7	Q, N	Polar uncharged
C_8	C	Sulphur containing, slightly charged
C_9	G	Smallest and the most flexible

D. Performance of K-VOGUE vs VOGUE vs HMMER on SCOP data

In this section we conducted experiments on the SCOP data set on VOGUE and K-VOGUE vs HMMER. We first collected clusters of the 20 amino acids from the expert in the field based on several chemical characteristics. Afterwards, we checked the efficiency of the clustering of the symbols by using the spectral clustering method previously described. The clusters of the 20

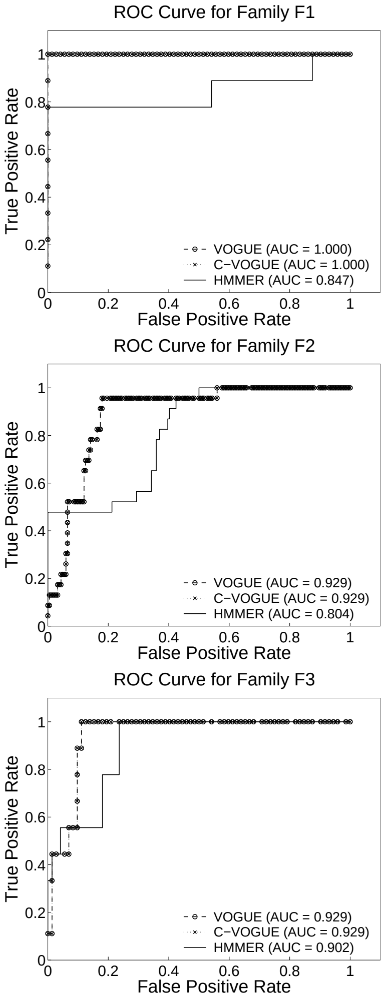


Fig. 10. ROC Curve of VOGUE, C-VOGUE and HMMER for the families F_1, F_2 and F_3.

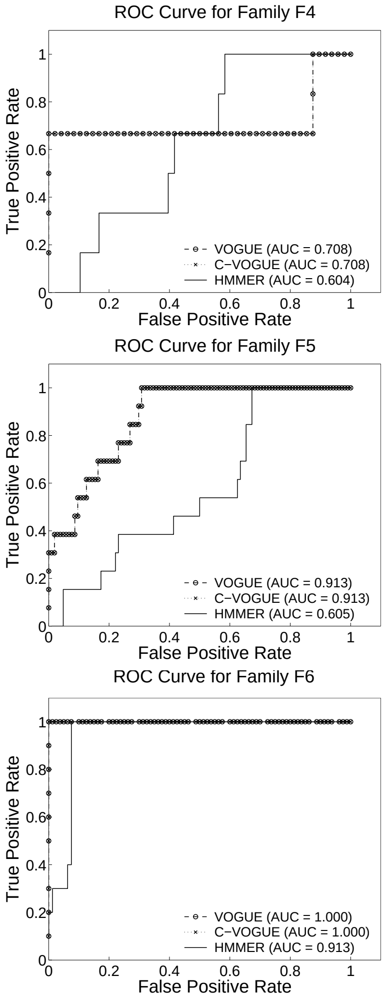


Fig. 11. ROC Curve of VOGUE, C-VOGUE and HMMER for the families F_4, F_5 and F_6.

TABLE XIX

AMINO ACIDS GROUPING

	Group 1	Group 2	Group 3
HYDROPHOBICITY	polar R K E D Q N	neutral G A S T P H Y	hydrophobic C V L I M F W
POLARITY	4.9 to 6.2 L I F W C M V Y	8.0 to 9.2 P A T G S	10.0 to 13.0 H Q R K N E D
POLARIZABILITY	0 to 0.108 G A S D T	0.128 to 0.186 C P N V E Q I L	0.219 to 0.409 K M H F R Y W
CHARGE	positive H R K	negative D E	other M F Y W C P N V Q I L N
NORMALIZED VAN DERWAALS VOLUME	0 to 2.8 G A S C T P D	2.95 to 4.0 N V E Q I L	4.43 to 8.08 M H K F R Y W

amino acids suggested by the expert were 9 cluster as given in Table XVIII.

In order to check the efficiency of the clustering of the symbols in case the expert's clustering is not available, we used the spectral clustering method described in Chapter 4. As domain knowledge we input the amino acids specifications which groups the amino acids in 3 groups based on the following five criteria:

1) **Hydrophobicity**: an amino acid can be either polar, neutral or hydrophobic.
2) **Polarity**: if the polarity of an amino acid ranges from 4.9 to 6.2 it belongs to the group 1, if it ranges from 8.0 to 9.2 it belongs to group 2 and if it ranges from 10.0 to 13.0 it belongs to group 3.
3) **Polarizability**: If the polarizability of the amino acid ranges from 0 to 0.108 it belongs to the group 1, if it ranges from 0.128 to 0.180 it belongs to group 2 and if it ranges from 0.210 to 0.409 it belongs to group 3.
4) **Charge**: an amino acid could be either positive, negative or carrying a small charge or no charge, called other.
5) **Normalized Van Derwaals volume**: if the volume of the amino acid ranges from 0 to 2.8 it belongs to the group 1, if it ranges from 2.95 to 4.0 it belongs to group 2 and if it ranges from 4.43 to 8.08 it belongs to group 3.

We used K-means with different values of the number of clusters $K(5, 6, 7, 9)$. The best results of the clustering was for $K = 9$ and is shown in Figure 13. The X axis represents the K eigenvectors that correspond to the K largest eigenvalues. The last column represents the cluster index in color to which the amino acid belongs to. The Y axis represents the amino acid index. The amino acids indexes are represented from 1 to 20. Clustering using the spectral clustering using eigenvectors and K-means with $K = 9$, we obtained the following 9 clusters as described in Table XX. Figure 14 shows the amino acid indexes sorted to group the amino acids belonging to the same cluster together. For example the last cluster whose color is dark red corresponds to the cluster $\mathcal{C}_1$ that contains the amino acids (H, K, R).

This clustering is very close to that of the expert. In fact, clusters $\mathcal{C}_1$, $\mathcal{C}_5$, $\mathcal{C}_7$, $\mathcal{C}_8$ were exact match with the experts clusters. Clusters $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_4$, and $\mathcal{C}_6$ were partially identified correctly. In fact, L and I were identified to belong to the same cluster $\mathcal{C}_2$. F, W were identified as belonging to the same cluster $\mathcal{C}_3$. S, T were identified to belong to the same cluster $\mathcal{C}_6$, while G, Y,

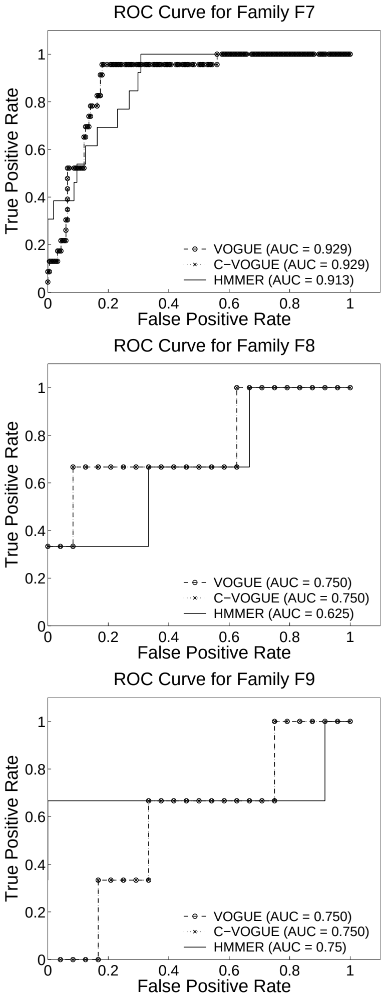


Fig. 12. ROC Curve of VOGUE, C-VOGUE and HMMER for the families F_7, F_8 and F_9.

A, M, and V were misclassified. Therefore,in the absence of the expert's clustering we can use cluster the alphabet $\sum$'s symbols using domain information (knowledge) and Spectral clustering as described in Chapter 4.

1) K-VOGUE vs HMMER Performance: Once we have the clusters of amino acids from the expert, we transform the data set by replacing the amino acids belonging to one cluster with the representative of that class. For example, for the cluster $\mathcal{C}_1 = \{H, K, R\}$, we replace any instance of K, R, H in the data set with the representative of the class which is H. We built K-VOGUE state machines with different values of $minsup$ corresponding to 50%, 75% and 100% of the number of instances in the training data, and $maxgap$ (10, 15, 20, 25, 30) but with the constant $k = 2$ for the length of the mined sequences in VGS. We then choose the best set of parameters and fix them for the remaining experiments. To model the data using HMMER, we first align the training sequences using CLUSTAL-W (`http://www.ebi.ac.uk/clustalw`). We then build a profile HMM using the multiple sequence alignment and compute the scores for each test sequence using HMMER, which directly reports the log-odds scores with respect to the **Null** model mentioned above.

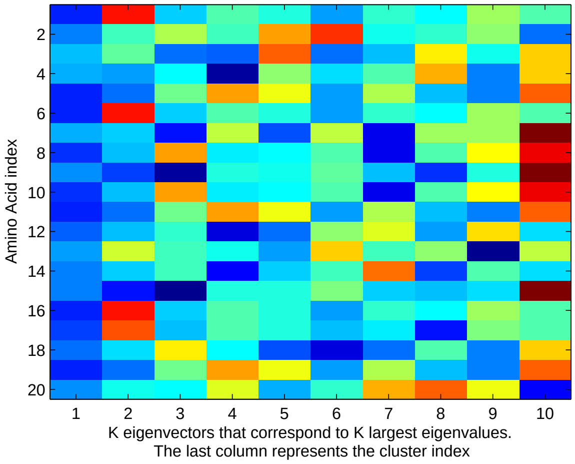


Fig. 13. Eigenvectors before sorting with $K = 9$ and using the Amino Acids Grouping.

TABLE XX

THE 9 CLUSTERS FOR THE AMINO ACIDS FROM K-MEANS CLUSTERING

Cluster	Elements	Indexes of elements
$\mathcal{C}_1$	H, K, R	$(7, 9, 15)$
$\mathcal{C}_2$	I, L	$(8, 10)$
$\mathcal{C}_3$	F, M, W	$(5, 11, 19)$
$\mathcal{C}_4$	D, E, V	$(3, 4, 18)$
$\mathcal{C}_5$	P	(13)
$\mathcal{C}_6$	A, G, S, T	$(1, 6, 16, 17)$
$\mathcal{C}_7$	N, Q	$(12, 14)$
$\mathcal{C}_8$	C	(2)
$\mathcal{C}_9$	(Y)	(20)

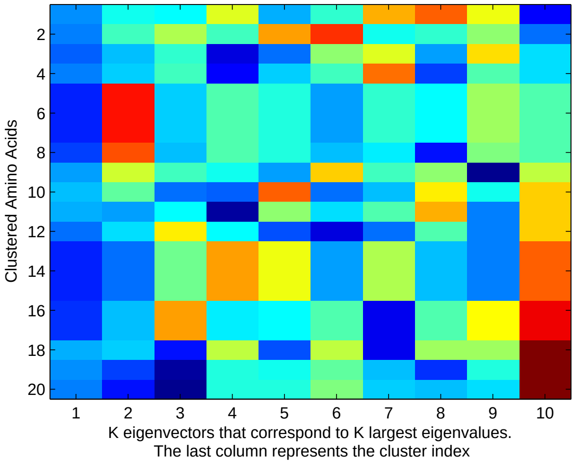


Fig. 14. Eigenvectors after sorting with $K = 9$ and using the Amino Acids Grouping.

Score Comparison: We first compare K-VOGUE HMMER. Table XXI shows the comparison on the 10 testing sequences from all the 10 families when scored against the model for families SF_2 and SF_5. For K-VOGUE we used $minsup = 6(75\%)$ and $maxgap = 20$. The best score for each sequence is highlighted in bold. Note that a negative score mean does not belong to the family and that a positive score means that it belongs to the family. Therefore, a very low negative score for sequence S_2 on model of family SF_{10} means it does not belong. Thus the smaller that score it is the better it is. Looking at the scores in Table XXI, we find that in general HMMER did well since it classified all the sequences as not belonging to family SF_2 (all scores were negative). However, sequence S_2 should have a positive score (S_2 belonging to SF_2) but it has a negative score of -4.20. Moreover, for all the 10 testing sequences K-VOGUE vastly outperforms HMMER for family SF_2. All the scores by

TABLE XXI

TEST SEQUENCE LOG-ODDS SCORES FOR K-VOGUE AND HMMER

Seq	HMMER SF_2	K-VOGUE SF_2	HMMER SF_5	K-VOGUE SF_5
S_1	-87.20	$\mathbf{-4760.38}$	-166.70	$\mathbf{-4,286.09}$
S_2	-4.20	$\mathbf{537.62}$	-141.60	$\mathbf{-1,896.44}$
S_3	-44.70	$\mathbf{-1709.69}$	-76.70	$\mathbf{-4,523.37}$
S_4	-31.60	$\mathbf{-1903.78}$	-74.50	$\mathbf{-4,525.66}$
S_5	$\mathbf{-32.50}$	-1.42	95.80	$\mathbf{197.39}$
S_6	-46.70	$\mathbf{-259.53}$	37.70	$\mathbf{-20.37}$
S_7	-89.50	$\mathbf{-2210.78}$	-140.60	$\mathbf{-220.44}$
S_8	-268.60	$\mathbf{-704.71}$	-348.80	$\mathbf{-13,520.02}$
S_9	-108.40	$\mathbf{-940.51}$	-179.00	$\mathbf{-4,642.94}$
S_{10}	-66.20	$\mathbf{-1764.22}$	-113.90	$\mathbf{-6,367.38}$

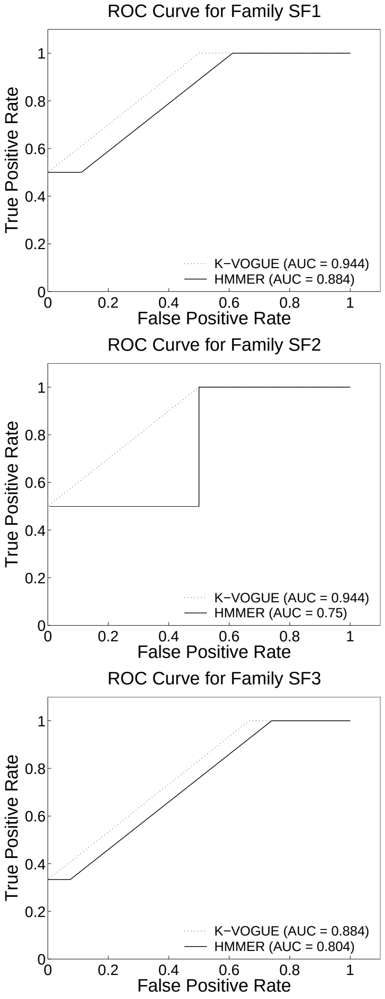


Fig. 15. ROC Curve of K-VOGUE and HMMER for the families SF_1, SF_2 and SF_3.

K-VOGUE were better than those of HMMER in this case, except for sequence S_5. The score of S_5 (-1.42) was higher than that of HMMER (-32.50) but still negative classifying S_5 as not belonging to SF_2. Concerning family SF_5, K-VOGUE again outperformed HMMER, since it only classified S_5 as belonging to SF_5 but the remaining of the testing sequences not belonging. HMMER classified correctly all the sequences but sequence S_6. With a score of 37.70, sequence S_6 was classified as belonging to family SF_5.

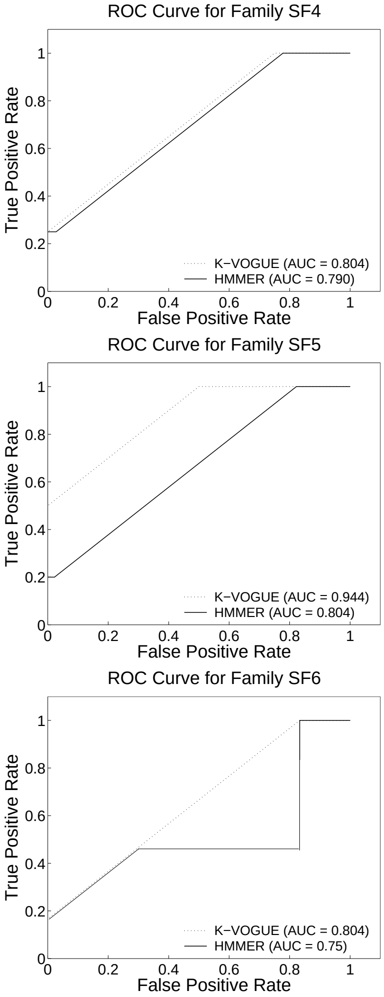


Fig. 16. ROC Curve of K-VOGUE and HMMER for the families SF_4, SF_5 and SF_6.

Full Comparison (ROC Curves and AUC): Figures 15 and 16 present the ROC curves of 6 families generated from all the testing sequences. Here we focus on comparing HMMER and K-VOGUE. A good classifier will be located at the top left corner of the ROC graph, illustrating that this classifier has high coverage of true positives with few false positives. A trivial rejector will be at the bottom left corner of the ROC graph and a trivial acceptor will be at the top right corner of the graph. Each one of the graphs in Figures 15, 16 has two ROC curves for K-VOGUE and HMMER, respectively, for different threshold values. The total AUC for the two methods is given in the legend. K-VOGUE was run with parameter typical values of $minsup = 75\%$ and $maxgap = 20$; there were some minor variations to account for characteristics of different families. The ROC curves of all the families show clearly that VOGUE improved the classification of the data over HMMER because the AUC of VOGUE is constantly higher than HMMER.

VIII. CONCLUSIONS AND FUTURE WORK

We introduced a new state machine called VOGUE to discover and interpret temporal dependencies in the analyzed data. We formally defined the two steps of this technique, where the first step uses a new and efficient sequence mining algorithm, Variable-Gap Sequence mining (VGS), to mine frequent patterns, and the second step uses these mined sequences to build VOGUE.

An important contribution of our new technique is that we are able to *simultaneously model* multiple higher-order HMMs due to the inclusion of variable length gaps allowed in the mined sequences. Once the model is built, it can be used to interpret new observation sequences. Therfore, we modified the widely used Viterbi algorithm into VG-Viterbi, to take into consideration the special topology of VOGUE. We showed experimentally, using real protein sequence data, that VOGUE's modeling power is superior to higher-order HMMs, as well as a domain-specific algorithm HMMER.

We further generalized VOGUE to any length, $k \geq 2$, of the sequences mined by VGS. Furthermore, some patterns mined by VGS are artifacts of other patterns, for example, if $A \rightarrow B$ is frequent, then there is a good chance that $B \rightarrow A$ will be frequent as well. We developed a special pruning mechanism, called, C-VOGUE, to separate primary patterns from artifacts.We showed through experimental results that, C-VOGUE reduces further the state space complexity of VOGUE while maintaining a good coverage and accuracy. Moreover, there are applications where there isn't always an exact match for the subsequences to be mined, such as in bioinformatics or in user data access. We extended VOGUE to K-VOGUE, to allow for approximate matches for the mined sequences and states. K-VOGUE takes into consideration that some elements in the alphabet Σ share similar characteristics and hence are similar. These elements are clustered either by a domain expert or by using domain information and spectral clustering as clustering technique. Then, VGS looks for frequent patterns whose elements belong to the same cluster instead of an exact match between the elements.

We used pseudo-counts in the transition, emission and duration probabilities, to account for the symbols that were not present in the training data set but might be present in the testing data sets. The values of these pseudo-counts were heuristically chosen but they were fixed for all the symbols. We need to automate this process and allow for pseudo-count values that reflect the overall distribution of the symbols. In fact, a symbol might have higher occurrences than others and hence its pseudo-count should be higher. Moreover, we need to understand what is the impact of the chosen pseudo-count value on the performance of VOGUE.

Finally, we demonstrate that VOGUE and its variations were able to outperform the state-of-the-art techniques in biological sequence clustering and analysis. VOGUE can be further used in other applications such as user access behavior [2], web prefetching [8], security [75], and many more interesting and challenging real world problems.

REFERENCES

[1] K. Aas, L. Eikvil, and T. Andersen. Text recognition from grey level images using hidden Markov models. *Lecture Notes in Computer Science*, 970, 1995.

[2] S. Adali, B. Bouqata, A. Marcus, B. Szymanski, and F. Spear. A day in the life of a metamorphic petrologist. *22nd Int'l Conf. on Data Engineering*, 2006.

[3] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. *Fast discovery of association rules.* In Fayyad, U., and al. eds., Advances in Knowledge Discovery and Data Mining. AAAI Press, Menlo Park, CA, 1998.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *20th Intl. Conf. Very Large Databases (VLDB).*, September 1994.

[5] R. Agrawal and R. Srikant. Mining sequential patterns. *11th IDE Conf.*, 1995.

[6] C. Antunes and A. L. Oliveira. Generalization of pattern-growth methods for sequential pattern mining with gap constraints. *Machine Learning and Data Mining in Pattern Recognition, Springer LNCS Vol. 2734*, pages 239–251, 2003.

[7] S. F. Atschul. Amino acid substitution matrices from an information theoretics perspective. *Molecular Biology*, 219:555–665, 1991.

[8] Bouqata B., Carothers C., Szymanski B., and Zaki M. Understading filesystem performance for data mining applications. *Proceedings of the 6th International Workshop on High Performance Data Mining: Pervasive and Data Stream Mining (HPDM:PDS'03)*, 2003.

[9] P. Baldi, Y. Chauvin, T. Hunkapiller, and M. McClure. Hidden markov models of biological primary sequence information. *Proc. Nat'l Academy of Sciences*, 91:1059–1063, Feb. 1994.

[10] L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, pages 1–8, 1972.

[11] L. E. Baum and J. Egon. An inequality with applications to statistical estimation for probabilistic functions of markov process and to a model for ecology. *Bull. Amer. Meteorol. Soc.*, pages 360–363, 1967.

[12] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *Ann. Math. Stat.*, pages 1554–1563, 1966.

[13] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Ann. Math. Stat.*, (1):164–171, 1970.

[14] L. E. Baum and G. R. Sell. Growth functions for transformations on manifolds. *Pacif. J. Math.*, (2):211–227, 1968.

[15] G. Bernardi, B. Olofsson, J. Filipski, M. Zerial, J. Salinas, G. Curry, M. Meunier-Rotival, and F. Rodier. The mosaic genome of warm-blooded vertebrates. *Science*, 228(4702):953–958, 1985.

[16] C. Bettini, X.S. Wang, S. Jajodia, and J.L. Lin. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):222–237, March 1998.

[17] M. P. Brown, R. Hughey, A. Krogh, I. S. Mian, K. Sjölander, and D. Haussler. Using dirichlet mixture priors to derive hidden markov models for protein families. pages 47–55, 1993.

[18] M. Brudno, S. Malde, A. Poloakov, C. B. Do, O. Couronne, I. Dubchak, and S. Batzoglou. Global alignment finding rearrangements during alignment. *Bioinformatics*, 19(1):54–62, 2003.

[19] L. Lo Conte, S. E. Brenner, T. J. P. Hubbard, C. Chothia, and A. G. Murzin. Scop database in 2002: refinement accommodate structural genomics. *Nucleic Acids Res.*, 30:264–267, 2002.

[20] E. Coward and F. Drablos. Detecting periodic patterns in biological sequences. *Bioinformatics, Oxford university press*, 14:498–507, 1998.

[21] B. A. Davey and H. A. Priestley. Introduction to lattices and order. *Cambridge University Press*, 1990.

[22] M. Deshpande and G. Karypis. Selective markov models for predicting web-page accesses. April 2001.

[23] C. H. Q. Ding and I. Dubchak. Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics.*, 17:349–358, 2001.

[24] R. F. Doolittle. Redundancies in protein sequences. *In Predictions of Protein Structure and the Principles of Protein Conformation (Fasman, G.D. ed) Plenum Press, New York*, pages 599–623, 1989.

[25] J.A. du Preez. Efficient training of high-order hidden markov models using first-order representation. *Computer Speech and Language*, 12(1):23–39, 1998.

[26] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic nodels for proteins and nucleic acids.* Cambridge Univ. Press, 1998.

[27] R. Eddy. Where did the blosum62 alignment score matrix come from?. *Nat. Biotechnology*, 22(8):1035–6, 2004.

[28] S. R. Eddy. Hidden markov models. *Current Opinion in Structural Biology*, 6:361–365, 1996.

[29] S. R. Eddy. Profile hidden markov models. *Bioinformatics*, 14:755–763, 1998.

[30] H. A. Edelstein. *Introduction to data mining and knowledge discovery (3rd ed).* Two Crows Corp, Potomac, MD, 1999.

[31] E. S. Lander et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[32] J. C. Venter et al. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2002.

[33] P.F. Evangelista, M.J. Embrechts, P. Bonissone, and B. K. Szymanski. Fuzzy roc curves for unsupervised nonparametric ensemble techniques. *International Joint Conference on Neural Networks (IJCNN'05)*, 5:3040–3045, 2005.

[34] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in knowledge discovery and data mining.* MIT Press., Cambridge, MA, 1996.

[35] P. Felzenszwalb, D. Huttenlocher, and J. Kleinberg. Fast algorithms for large-state-space hmms with applications to web usage analysis. *In Advances in Neural Information Processing Systems*, 2003.

[36] P. G. Ferreira and P. J. Azevedo. Protein sequence classification through relevant sequence mining and bayes classifiers. *Springer-Verlag*, pages 236–247, 2005.

[37] J. W. Fickett and C.-S. Tung. Assessment of protein coding measures. *Nucleic Acids Research*, 20(24):6441–6450, 1992.

[38] G. D. Forney. The viterbi algorithm. *Proc. IEEE*, 61:268–278, March 1973.

[39] M. Garofalakis, R.Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. *In 25th Intl. Conf. VEry Large Databases*, 1999.

[40] M. Garofalakis, R.Rastogi, and K. Shim. Mining sequential patterns with regular expression constraints. *IEEE TKDE*, 14:530–552, May 2002.

[41] J. Han and M. Kamber. *Data mining: Concepts and Techniques.* Morgan-Kaufman, New York, NY, 2000.

[42] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning : Data mining, inference, and prediction.* Springer, New York, NY, 2001.

[43] K. Hatonen, M. Klemettinen, H. Mannila, P. Ronkainen, and H. Toivonen. Knowledge discovery from telecommunication network alarm databases. *12th Intl. Conf. Data Engineering*, February 1996.

[44] L. S. Ho and J. C. Rajapakse. Splice site detection with higher-order markov model implemented on a neural network. *Genome Informatics*, 14:64–72, 2003.

[45] M. Holsheimer, M. Kersten, H. Mannila, and H. Toivonen. A perspective on databases and data mining. *Proc. of 1st Intl. Conf. on knowledge Discovery and Data Mining (KDD)*, August 1995.

[46] R. Kannan, S. Vempala, and A. Vetta. On clusterings - good, bad, and spectral. *In Proc. of the 41st Annual Symposium on Foundations of Computer Science*, 2000.

[47] T. Kanungo. Stochastic language models for style-directed layout analysis of document images. *IEEE Transactions on Image Processing*, 12(5), May 2003.

[48] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. *In 3rd Intl. Conf. Information and Knowledge Management,*, pages 401–407, November 1994.

[49] S. Koenig, R. Goodwin, and R. Simmons. Robot navigation with markov models: A framework for path planning and learning with limited computational resources. December 1995.

[50] S. Kurtz, J. V. Choudhuri, E. Ohlebusch, C. Schleiermacher, J. Stoye, and R. Giegerich. Reputer: the manifold applications of repeat analysis on a genemic scale. *Nucleic Acids Research*, 29(22):4633–4642, 2001.

[51] S.E. Levinson, L. R. Rabiner, and M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *Bell Syst. Tech. Journal*, 62(4):1035–1074, 1983.

[52] C. Liebecq. *Biochemical Nomenclature and Related Documents. 2d Edition*. Portland Press, London, 1992.

[53] J. Lin, E.Keogh, S. Lonardi, and P. Patel. Finding motifs in time series. *Temporal Data Mining Workshop Notes and K.P. Unnikrishnan and R. Uthurusamy and eds*, July 2002.

[54] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurences. *2d Intl. Conf. Knowledge Discover and Data Mining*, 1996.

[55] H. Mannila, H. Toivonen, and I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery: An international Journal,*, 1(3):259–289, 1997.

[56] D. Meyer. Human Gait Classification Based on Hidden Markov Models. pages 139–146, 1997.

[57] D. M. Mount. Bioinformatics: Sequence and genome analysis 2nd ed. *Cold Spring Harbor Laboratory Press: Cold Spring Harbor, NY*, 2004.

[58] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized web prefetching. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1155–1169, 2003.

[59] A. Y. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *In Proc. of the NIPS-14*, 2001.

[60] R. T. Ng, L. V. S. Lakshmanan, and J. Han. Exploratory mining and pruning optimizations of constrained association rules. *In ACM SIGMOD Intl. Conf. Management of Data,*, June 1998.

[61] C. Notredame. Recent progresses in multiple sequence alignement: a survey. *Pharmacogenomics (2002)*, 3(1), 2002.

[62] T. Oates, M. D. Schmill, D Jensen, and P. R. Cohen. A family of algorithms for finding temporal structure in data. *In 6th Intl. Workshop on AI and Statistics,*, March 1997.

[63] J. S. Pedersen and J. Hein. Gene finding with a hidden markov model of genome structure and evolution. *Bioinfromatics*, 19(2):219–227, 2003.

[64] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M-C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix projected pattern growth. April 2001.

[65] P. Pirolli and J. E. Pitkow. Distribution of surfers paths through the world wide web: Empirical characterization. volume 2(1-2), pages 29–45, 1999.

[66] J. Pitkow and P. Pirolli. Mining longest repeating subsequence to predict WWW surfing. 1999.

[67] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings IEEE*, 2(77):257–286, 1989.

[68] P. S. Reddy and D. E. Housman. The complex pathology of trinucleotide repeats. *Current Opin. Cell Biol.*, 9:364–372, 1997.

[69] M.J. Russell and R. K. Moore. Explicit modeling of state occupancy in hidden markov models for automatic speech recognition. pages 5–8, 1985.

[70] J. Salazar, M. Robinson, and M. R. Azimi-Sadjadi. A hybrid hmm-neural network with gradient descent parameter training. *Proceedings IJCNN03*, 2003.

[71] L. Saul and M. Jordan. Mixed memory markov models: Decomposing complex stochastic processes as mix of simpler ones. *Machine Learning*, 37(1):75–87, 1999.

[72] L.C. Schwardt and J.A. du Preez. Efficient mixed-order hidden markov model inference. oct 2000.

[73] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000.

[74] R. Srikant and R. Agrawal. Mining sequential patterns: Generalization and performance improvements. *5th Int'l Conf. on Extending Database Technology*, 1996.

[75] B. Szymanski and Y. Zhang. Recursive data mining for masquerade detection and author identification. *Proc. 5th IEEE System, Man and Cybernetics Information Assurance Workshop*, pages 424–431, June 2004.

[76] J. D. Thopmson, D. G. Gibson Higgins, and T. J. Gibson. Clustalw: improving the sensitivity if progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22:4673–4680, 1994.

[77] A. van Belkum, S. Scherer, W. van Leeuwen, D. Willemse, L. van Alphen, and H. Verburgh. Variable number tandem repeats in clinical strains hemophilus influenzae. *Infection and Immunity*, 65(12):5017–5027, 1997.

[78] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–269, April 1967.

[79] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. Constrained k-means clustering with background knowledge. *In ICML*, 2001.

[80] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Comput. Biology*, 1:337–348, 1994.

[81] S. M. Weiss and N. Indurkhya. *Predictive data mining: A practical guide.* Morgan-Kaufman, New York, NY, 1997.

[82] C. Westphal and T. Blaxton. *Data mining solutions.* Wiley, New York, NY, 1998.

[83] I. H. Witten and E. Frank. *Data mining.* Morgan-Kaufmann, New York, NY, 2000.

[84] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. *In NIPS-15*, 2003.

[85] T. Yang, T. Mitra, and T-C. Chiueh. A decoupled architecture for application-specific file prefetching. 2002.

[86] S. X. Yu and J. Shi. Grouping with bias. *In NIPS-14*, 2002.

[87] S. X. Yu and J. Shi. Multiclass spectral clustering. *In International Conference on Computer Vision*, 2003.

[88] M. J. Zaki. Sequences mining in categorical domains: Incorporating constraints. November 2000.

[89] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42(1/2):31–60, Jan/Feb 2001.

[90] M. J. Zaki, N. Lesh, and M. Ogihara. Planmine: Sequence mining for plan failures. *4th Intl. Conf. Knowledge Discovery and Data Mining*, August 1998.

[91] H. Zha, C. Ding, M. Gu, X. He, and H. Simon. Spectral relaxation for k-means clustering. *In NIPS-14*, 2002.

[92] M. Zhang, B. Kao, D. W. Cheung, and K. Y. Yip. Mining periodic patterns with gap requirement from sequences. *SIGMOD*, pages 623–633, June 2005.

[93] S. Zoubak, O. Clay, and G. Bernardi. The gene distribution of the human genome. *Gene*, 174:95–102, 1996.