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

Bouchra Bouqata, Christopher D. Carothers,
Boleslaw K. Szymanski, and Mohammed J. Zaki

CS Department, Rensselaer Polytechnic Institute, Troy, NY, USA
{bouqab, chrisc, szymansk, zaki}@cs.rpi.edu

Abstract. We present VOGUE, a new state machine that combines two separate techniques for modeling long range dependencies in sequential data: data mining and data modeling. VOGUE relies on a novel Variable-Gap Sequence mining method (VGS), to mine frequent patterns with different lengths and gaps between elements. It then uses these mined sequences to build the state machine. We applied VOGUE to the task of protein sequence classification on real data from the PROSITE protein families. We show that VOGUE yields significantly better scores than higher-order Hidden Markov Models. Moreover, we show that VOGUE's classification sensitivity outperforms that of HMMER, a state-of-the-art method for protein classification.

1 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: sequence pattern mining [12], and data modeling via Hidden Markov Models (HMMs) [4, 8]. 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 [8–10] have been proposed, where the order denotes the number of previous states that the current state depends upon. However higher-order HMMs often suffer from a number of difficulties, namely, *high state-space complexity*, *reduced coverage*, and sometimes even *low prediction accuracy* [3].

In this paper we present a new approach to temporal/sequential data analysis via a novel state machine, **VOGUE** (**V**ariable **O**rder **G**aps for **U**nstructured **E**lements). The first step of our method uses a new sequence mining algorithm, called **V**ariable-**G**ap **S**equence miner (VGS), to mine variable-length frequent patterns, that may contain different gaps between the elements. The second step of our technique uses the mined variable-gap sequences to build the VOGUE state machine. In fact, VOGUE models multiple higher order HMMs via a single variable-order state machine. Although VOGUE has a much wider applicability,

J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (Eds.): PKDD 2006, LNAI 4213, pp. 42–54, 2006.
© Springer-Verlag Berlin Heidelberg 2006

in this paper we apply it to a 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 [4] 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 capabilities. VOGUE also outperforms HMMER [4], a HMM model especially designed for protein sequences.

2 Related Work

HMMs have been proposed to model longer range dependencies. However, such models suffer from high state-space complexity, since a k-th order HMM, with alphabet Σ, can potentially have $|\Sigma|^k$ states. Estimating the joint probabilities of each k-th order state is also difficult. The all-k-order Markov model was proposed in [8], where one has to maintain a Markov model of order j (where the current state depends on the j previous states) **for all** $1 \leq j \leq k$. Three *post-pruning* techniques were proposed in [3] to improve the prediction accuracy and coverage, and to lower the state complexity of the all k-order Markov model. However, multiple models still have to be maintained.

In [9], mixed order Markov models were proposed. However, they rely on Expectation Maximization (EM) algorithms that are prone to local optima. Furthermore, mixed order Markov models depend on a mixture of bigrams over k consecutive previous states, whereas VOGUE automatically ignores irrelevant states. Another approach combines the mining of sequences with a Markov predictor for web prefetching [7], but it is tuned specifically for web usage mining since it relies on the knowledge of the site structure. In [10], a suffix tree is incorporated in the training of the HMM which is done by an EM algorithm. Although this algorithm reduces the state space complexity of an all-k-order HMM, it still uses the previous k states and relies on an EM method. The Episode Generating HMM (EGH) [6] is especially relevant. However, there are notable differences in the EGH approach versus VOGUE. Instead of building EGHs per subsequence, VOGUE is a *single* variable-order state machine incorporating all the frequent sequences. In VOGUE, the gap states have the notion of duration which enables our system to account for long range dependencies. The Hierarchical HMM (HHMM) approach in [2] extracts episodes (using sequence alignment methods) from which (left-to-right) HMMs are built. HHMM also emits a random or "any" symbol in a gap state. In contrast, VOGUE simultaneously models all non-consecutive patterns, as well as gap symbol and duration statistics.

3 VOGUE State Machine

As noted earlier, building higher order HMMs is not easy, since we have to estimate the joint probabilities of the previous k states in a *k-order* HMM. Also, not all of the previous k 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 an EM (also known as Baum-Welch) algorithm for training the model. VOGUE addresses these limitations. It first uses the VGS algorithm to mine variable-gap frequent sequences that can have g other symbols between any two elements; g varies from 0 to a maximum gap ($MAXGAP$). These sequences are then used as the estimates of the joint probabilities for the states used to seed the model.

Consider a simple example to illustrate our main idea. Let the alphabet be $\Sigma = \{A, \cdots, K\}$ and the sequence be $S = \mathbf{ABA}C\mathbf{B}D\mathbf{A}EF\mathbf{B}GH\mathbf{A}IJK\mathbf{B}$. 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 first-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: 1) **Pattern Mining** via the novel Variable-Gap Sequence (VGS) mining algorithm. 2) **Data Modeling** via our novel Variable-Order state machine. 3) **Interpretation** of new data via a modified Viterbi method, called VG-Viterbi, to model the most probable path through a VOGUE model. Details of these steps appear below.

3.1 Mining: Variable-Gap Sequences (VGS)

VGS is based on cSPADE [11, 12], a method for constrained sequence mining. Whereas cSPADE essentially ignores the length of and symbol distributions in gaps, VGS is specially designed to extract such patterns within one or more sequences. Note that whereas other methods can also mine gapped sequences [1, 11], 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 ($minsup$). VGS mines all sequences having up to k elements, with a gap of at most $maxgap$ length between any two elements, such that the sequence occurs at least $minsup$ 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 1. Those symbols that are frequent are extended to consider sequences of length 2, as shown in Table 2. 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 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 have been mined.

Table 1. 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 2. 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

3.2 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 sequences 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 and lack of space we will illustrate the working of VOGUE using mined sequences of length $k = 2$. 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 $s_i : v_f \rightarrow v_s$, where $v_f, v_s \in \Sigma$. Let $\Gamma = \{v_f | v_f \rightarrow v_s \in F_2\}$ be the set of all the distinct symbols in the first position, and $\Theta = \{v_s | v_f \rightarrow v_s \in F_2\}$ 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 $\Sigma = \{v_1, \cdots v_M\}$, where $|\Sigma| = M$ is the number of observations emitted over all states. The alphabet's size is

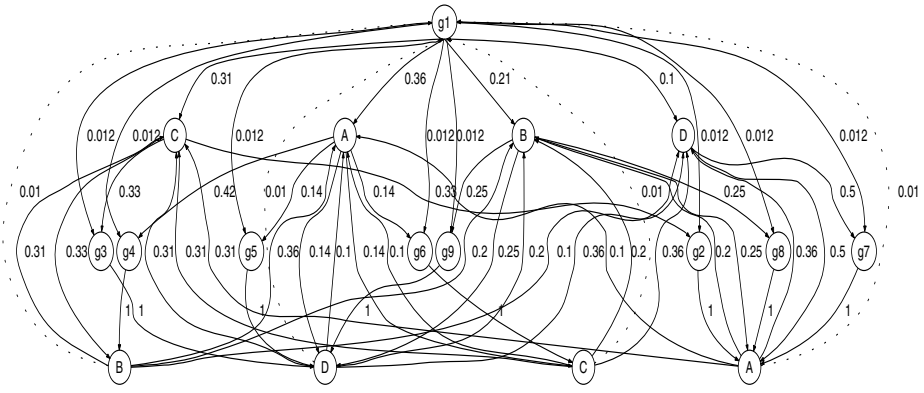

Fig. 1. VOGUE State Machine for Running Example

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 1. For our example S in Section 3.1, we have nine distinct frequent symbols, thus $M = 9$.

Set of States (Q): The set of states in VOGUE is given as $Q = \{q_1, \cdots, q_N\}$, where $|Q| = N = N_f + G_i + N_s + G_u$. 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 $Q = Q_f \cup Q_i \cup Q_s \cup Q_u$ be the partition of Q 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 3.1, we have $N_f = 4$, since there are four distinct starting symbols in Table 2 (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 $G_i = 8$, $C \rightarrow B$ 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, $A = \{a(q_i, q_j) | 1 \leq i, j \leq N\}$, where $a(q_i, q_j) = P(q^{t+1} = q_j | q^t = q_i)$, 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 $s_{iy} : v_i \rightarrow v_y \in F_2$ 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., $freq_i(y) = \sum_{g=0}^{maxgap} freq_i^g(y)$. Let $R = \frac{freq_i^0(j)}{\sum_{y \in Q_s} freq_i(y)}$ denote the fraction of gap-less transitions from q_i to q_j over all the transitions from q_i to $q_y \in Q_s$. The transition probabilities from $q_i \in Q_f$ are given as:

$$a(q_i, q_j) = \begin{cases} R & \text{for } q_j \in Q_s \\ \frac{freq_i(j)}{\sum_{y \in Q_s} freq_i(y)} - R & \text{for } q_j \in Q_i \\ 0 & \text{for } q_j \in Q_f \cup Q_u \end{cases}$$

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:

$$a(q_i, q_j) = \begin{cases} 1 & \text{for } q_j \in Q_s \\ 0 & \text{for } q_j \in Q \backslash Q_s \end{cases}$$

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:

$$a(q_i, q_j) = \begin{cases} 0.99 \times \frac{\sum_{q_y \in Q_f} freq_j(y)}{T} & \text{for } q_j \in Q_f \\ 0.01 & \text{for } q_j \in Q_u \\ 0 & \text{for } q_j \in Q_i \cup Q_s \end{cases}$$

Note that the transitions to universal gap have a small probability (0.01). 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 the values 0.99 and 0.01 above were chosen to allow (via pseudo-counts) for unseen symbols.

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:

$$a(q_i, q_j) = \begin{cases} 0.9 \times \frac{\sum_{q_y \in Q_f} freq_j(y)}{T} & \text{for } q_j \in Q_f \\ 0.1 \times \frac{1}{G_i} & \text{for } q_j \in Q_i \\ 0 & \text{for } q_j \in Q \backslash Q_f \end{cases}$$

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. 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 1 shows transitions between states and their probabilities in VOGUE for our running example. Each gap state's duration is considered explicitly within a state. The notation g_i (e.g., g_3) is the name of the gap state between the elements of the sequence, (e.g., $C \rightarrow D$), 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: $B = \{b(q_i, v_m) = P(v_m|q_i), 1 \leq i \leq N \text{ and } 1 \leq m \leq M\}$, where $b(q_i, v_m) = P(v_m|q_i)$ 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 test data, we assign them a very small probability of emission in the gap states.

Non-gap States: If $q_i \in Q_f \cup Q_s$, then $b(q_i, v_m) = 1$ for the distinct symbol that can be emitted for that state, and $b(q_i, v_m) = 0$, otherwise. For example, the first and second states are labeled by their emission symbol in Figure 1.

Universal Gap: For $q_i \in Q_u$ we have $b(q_i, v_m) = \left(\frac{freq(v_m)}{\sum_{v_m \in \Sigma} freq(v_m)}\right) \times 0.99 + c'$, where $c' = 0.01/M$. 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 $b(q_i, v_m) = \left(\frac{\sum_{g \geq 1} freq_g(v_m, q_i)}{\sum_{v_m \in \Sigma_{q_i}} \sum_{g \geq 1} freq_g(v_m, q_i)}\right) \times 0.99 + c$, where $c = 0.01/|\Sigma_{q_i}|$.

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 3.1). Thus C's emission probability is $\frac{2}{5} \times 0.99 + \frac{0.01}{4} = 0.399$.

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: $\rho = \{\rho(q_i, g)|q_i \in Q_i, g \in [1, maxgap]\}$. 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 $s : v_x \rightarrow v_y \in F_2$. The gap duration probability is proportional to the frequency of observing a given gap value for s, i.e., $\rho(q_i, g) = \frac{freq_i^g(y)}{\sum_{g \in [1, maxgap]} freq_i^g(y)}$; and $\rho(q_i, g) = 1$ for $q_i \in Q \backslash Q_i$. 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 twice observe 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

$$\pi(i) = \begin{cases} 0.99 \times \frac{\sum_{q_y \in Q_f} freq_i(y)}{T} & \text{for } q_i \in Q_f \\ 0.01 & \text{for } q_i \in Q_u \\ 0 & \text{for } q_i \in Q_i \cup Q_s \end{cases}$$

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.

3.3 Interpretation: Variable-Gap Viterbi

Once VOGUE is built, given a new test sequence of observations $O = o_1 o_2 \cdots o_T$, there is a need to interpret the sequence given the model. This problem is equivalent to finding the best sequence of states, i.e., the *most probable path*, through the VOGUE model λ, for the test sequence O. That is finding a sequence of states $\mathbf{q}_* = \{q_*^1, q_*^2, \cdots, q_*^T\}$ from the model λ such that: $\mathbf{q}* = \arg\max_{\mathbf{q}} P(\mathbf{q}|\lambda, O)$, over all such sequence of states $\mathbf{q}$. The algorithm that is most often used to solve this problem for biosequences, is the Viterbi algorithm [4]. Due to the unique structure of VOGUE, where gap states have a notion of duration, we adjusted Viterbi to take this into account. We call our new method Variable-Gap Viterbi (VG-Viterbi). For the lack of space, we omit the algorithmic details of VG-Viterbi.

4 Experimental Results and Analysis

In recent years, a large amount of work in biological sequence analysis has focused on methods for 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. HMMER [4], 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 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 $k \in [1, 10]$.

Dataset: The data used in our experiments is a set of 9 families downloaded from the PROSITE (http://www.expasy.org/prosite) database of protein family and domains, namely, $PDOC00662$, $PDOC00670$, $PDOC00561$, $PDOC00064$, $PDOC00154$, $PDOC00224$, $PDOC00271$, $PDOC00397$, $PDOC00443$. 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 consist 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 test data N^i_{test} contains the remaining 10%. For example, $N^1_{train} = 40$ and $N^1_{test} = 5$. There are a total of 103 test sequences across all families.

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 [5], 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 [5]: AUC $= \frac{1}{pn} \sum_{i=1}^{p} \sum_{j=1}^{n} \varphi(R_i, R_j)$. Here $N_{test} = n + p$ is the number of test 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 test 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 $\varphi(R_i, R_j) = 1$if $R_i < R_j$, or else $\varphi(R_i, R_j) = 0$. AUC for each class is calculated separately, by treating each class as p, and the remaining as n.

We score the test sequences by computing the log-odds score, i.e., the ratio of the probability of the sequence using a given model, to the probability of the sequence using a **Null** model, given as follows: Log-Odds(seq) $= \log_2 \left(\frac{P(seq/Model)}{P(seq/\mathbf{Null})} \right)$. $P(seq/Model)$ 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.

4.1 Comparing VOGUE, HMMER and k-th Order HMMs

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. For 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 test sets were used as observation symbols. Then we model the resulting sequence with a regular 1st order HMM.

Table 3. Test Sequence Log-Odds Scores for VOGUE, HMMER and k-th Order HMMs

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

Score Comparison: We first compare VOGUE with k-order HMMs and HMMER. Table 3 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-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, respectively, $k = 2; 4; 8; 10$. These values were obtained from a count of the different new symbols used, as described earlier, for each value of k. The best score for each sequence is highlighted in bold. In Table 3, we find that 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 sequences. However, for all the 5 test 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 4, we show the execution times for family F_1. The time for VOGUE includes the mining by VGS, and for HMMER, the alignment by CLUSTAL-W. 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, but it also executes faster.

Table 4. Run Times

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

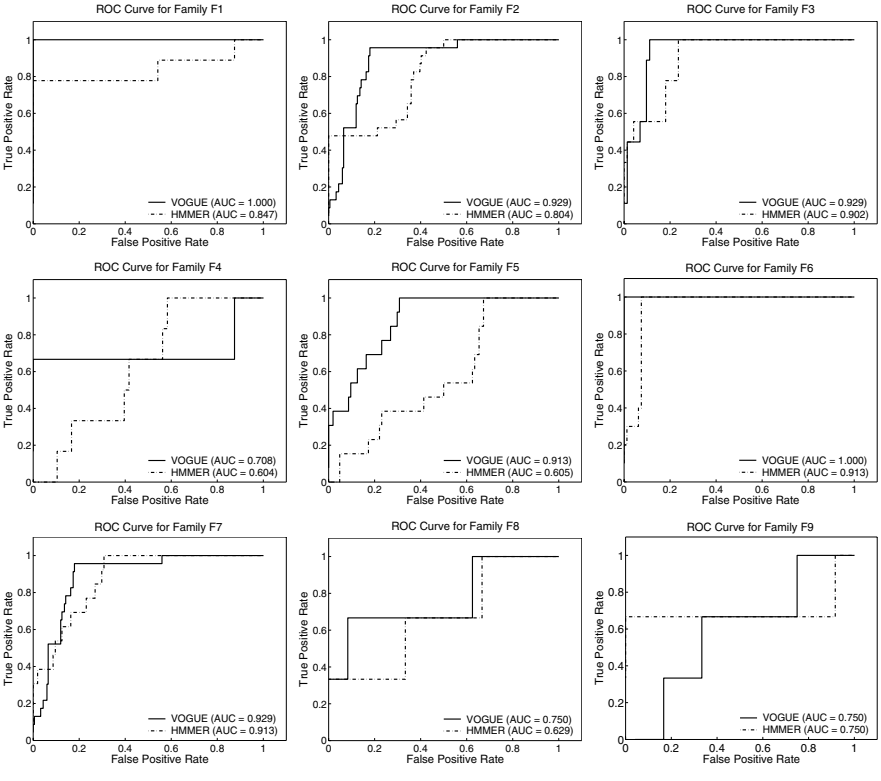

Fig. 2. ROC Curve of VOGUE and HMMER for the 9 families

Full Comparison (ROC Curves and AUC): Figure 2 presents the ROC curves of the 9 families generated from all the test sequences. Here we focus on comparing HMMER and VOGUE, since k-th order HMMs gave highly negative scores for all the test sequences. The ROC curves represent the trade-off between coverage (TPR on the y-axis) and error rate (FPR on the x-axis) of a classifier.

A good classifier will be located at the top left corner of the ROC graph. 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 Figure 2 has two ROC curves for VOGUE and HMMER, respectively, for different threshold values. The total AUC for the two methods is given in the legend. 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. 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.

5 Conclusions and Future Work

One of the main contribution of VOGUE is that it can *simultaneously model* multiple higher-order HMMs. We showed experimentally on protein sequence data that VOGUE's modeling power is superior to higher-order HMMs, as well as a domain-specific algorithm HMMER. To generalize VOGUE for sequences of $k > 2$ (after VGS), a special topology will be needed to handle interleaving patterns. 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 need special pruning mechanisms to separate primary patterns from artifacts. Moreover, there are applications where there is not always an exact match for the subsequences to be mined. In future work we plan to allow for approximate matches for the mined sequences and states.

Acknowledgments. This work was supported in part by NSF CAREER Award IIS-0092978, and NSF grant EIA-0103708.

References

1. 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.
2. M. Botta, U. Galassi and A. Giordana. Learning Complex and Sparse Events in Long Sequences. *European Conference on Artificial Intelligence*, 2004.
3. M. Deshpande and G. Karypis. Selective markov models for predicting web-page accesses. In *SIAM International Conference on Data Mining*, April 2001.
4. S. R. Eddy. Profile hidden markov models. *Bioinformatics*, 14:755–763, 1998.
5. P.F. Evangelista, M.J. Embrechts, P. Bonissone, and B. K. Szymanski. Fuzzy ROC curves for unsupervised nonparametric ensemble techniques. *IJCNN*, 2005.

6. S. Laxman, et al. Discovering frequent episodes and learning hidden markov models: A formal connection. *IEEE TKDE*, 17(11):1505–1517, Nov 2005.
7. A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized web prefetching. *IEEE TKDE*, 15(5):1155–1169, 2003.
8. J. Pitkow and P. Pirolli. Mining longest repeating subsequence to predict WWW surfing. In *2nd USENIX Symp. on Internet Technologies and Systems*, 1999.
9. 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.
10. L.C. Schwardt and J.A. du Preez. Efficient mixed-order hidden markov model inference. In *Int'l Conf. on Spoken Language Processing*, oct 2000.
11. M. J. Zaki. Sequences mining in categorical domains: Incorporating constraints. In *9th Int'l Conf. on Information and Knowledge Management*, November 2000.
12. M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42(1/2):31–60, Jan/Feb 2001.