

# VOGUE: A Variable Order Hidden Markov Model with Duration based on Frequent Sequence Mining

MOHAMMED J. ZAKI  
CHRISTOPHER D. CAROTHERS  
and  
BOLESŁAW K. SZYMANSKI  
Computer Science Department  
Rensselaer Polytechnic Institute  
{zaki,chrisc,szymansk}@cs.rpi.edu

---

We present VOGUE, a novel, variable order hidden Markov model with state durations, that combines two separate techniques for modeling complex patterns in sequential data: pattern mining and data modeling. VOGUE relies on a variable gap sequence mining method to extract frequent patterns with different lengths and gaps between elements. It then uses these mined sequences to build a variable order hidden Markov model, that explicitly models the gaps. The gaps implicitly model the order of the HMM, and they explicitly model the duration of each state. We apply VOGUE to a variety of real sequence data taken from domains such as protein sequence classification, web usage logs, intrusion detection, and spelling correction. We show that VOGUE has superior classification accuracy compared to regular HMMs, higher-order HMMs, and even special purpose HMMs like HMMER, which is a state-of-the-art method for protein classification. The VOGUE implementation and the datasets used in this paper are available as open-source at: [www.cs.rpi.edu/~zaki/software/VOGUE](http://www.cs.rpi.edu/~zaki/software/VOGUE).

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database applications—*Data Mining*; I.2.6 [**Artificial Intelligence**]: Learning; I.5.1 [**Pattern Recognition**]: Models; G.3 [**Probability and Statistics**]: —*Markov processes*

General Terms: Algorithms

Additional Key Words and Phrases: Hidden Markov Models, Higher-order HMM, HMM with Duration, Sequence Mining and Modeling, Variable-order HMM

---

## 1. INTRODUCTION

Many real world applications, such as those in bioinformatics, web accesses, and text mining, encompass sequential or temporal data with long and short range dependencies. Techniques for analyzing such data can be grouped in two broad categories: pattern mining and data modeling. Efficient methods have been pro-

---

Author's address: Mohammed J. Zaki, Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 2009 ACM 0000-0000/2009/0000-0001 \$5.00

posed for sequence pattern mining in both data mining [Srikant and Agrawal 1996; Mannila et al. 1995; Zaki 2001; Pei et al. 2001] and bioinformatics [Gusfield 1997; Jensen et al. 2006]. For sequence data modeling, Hidden Markov Models (HMMs) [Rabiner 1989] have been widely employed in a broad range of applications such as speech recognition, web usage analysis, and biological sequence analysis [Durbin et al. 1998; Felzenszwalb et al. 2003; Pitkow and Pirolli 1999].

A simple, or first-order, Markov model is a stochastic process where each state depends only on the previous state. In regular Markov models, each state emits only one symbol, and only the transition probabilities have to be estimated. A first-order hidden Markov model, on the other hand has “hidden” states, i.e., it may emit more than one symbol, according to the state-dependent output probabilities, which become additional parameters that have to be estimated. Due to their very nature first-order (hidden) Markov models may not suitably capture longer range sequential dependencies. For such problems, higher-order and variable-order Markov models [Pitkow and Pirolli 1999; Saul and Jordan 1999; Deshpande and Karypis 2001; Ron et al. 1996; Buhlmann and Wyner 1999] and HMMs [du Preez 1998; Schwardt and Du Preez 2000; Galassi et al. 2007; Law and C.Chan 1996; Wang et al. 2006] have been proposed. However, building an  $m$ -order HMM requires estimation of the joint probabilities of the previous  $m$  states. Furthermore, not all of the previous  $m$  states may be predictive of the current state. Hence, the training process is extremely expensive and suffers from local optima. As a result, higher-order models may suffer from *high state-space complexity*, *reduced coverage*, and sometimes even *low prediction accuracy* [Deshpande and Karypis 2001].

To address these limitations, we introduce a new approach to temporal/sequential data analysis that combines temporal pattern mining and data modeling. The rationale behind our approach is that many real-world sequences are difficult to directly model via HMMs due to the complex embedded dependencies. Instead we first extract these frequent sequential patterns via sequence mining, and then use those to construct the HMM. We introduce a new variable order HMM called **VOGUE** (**V**ariable **O**rders and **G**apped HMM for **U**nstructured **E**lements) to discover and interpret long and short range temporal locality and dependencies in the analyzed data <sup>1</sup>. The first step of our method uses a new sequence mining algorithm, called **V**ariable-**G**ap **S**equences 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 automatically build the variable-order VOGUE HMM, i.e., the topology of the model is learned directly from the mined sequences. In fact, VOGUE models multiple higher order HMMs via a single variable-order model with explicit duration.

In this paper we apply VOGUE on several real datasets. The first application is for 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 that 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 [Eddy 1998] have been developed to find homologs. However, in most biological sequences, interesting patterns repeat (either

<sup>1</sup>A preliminary version of this work appeared in [Bouqata et al. 2006]

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 outperforms HMMER [Eddy 1998], a HMM model specifically designed for protein sequences.

We also apply VOGUE to the problems of web usage mining, where the task is to distinguish the affiliation of the users, i.e., to find out whether users come from an academic (`edu`) or a commercial or other domain. We also looked at the problem of masquerading within intrusion detection. Given user command logs at a terminal we look at the task of identifying the user. Finally, we look at spelling correction. Given a dataset of commonly misspelt words along with their context, we determine whether the usage is correct or incorrect. It is important to keep in mind that the real purpose of using an HMM based approach like VOGUE is to model the data, i.e., to build a generative model. VOGUE can also be used as a discriminative model, but this mode is used mainly for comparison against other HMM methods.

## 2. RELATED WORK

A recent book [Dong and Pei 2007] provides a very good introduction to the issues that span sequence mining and modeling. Here we briefly review the most relevant research.

### 2.1 Sequence Mining

Sequence mining helps to discover frequent sequential patterns across time or positions in a given data set. Most of the methods in bioinformatics are designed to mine consecutive subsequences, such as those that rely on suffix trees [Gusfield 1997]. Within data mining, the problem of mining sequential patterns was introduced in [Agrawal and Srikant 1995]. The same authors then extended the approach to include constraints like max/min gaps, and taxonomies in [Srikant and Agrawal 1996]. Other sequence mining methods have also been proposed, such as SPADE [Zaki 2001], which was later extended to cSPADE [Zaki 2000] to consider constraints like max/min gaps and sliding windows. GenPrefixSpan [Antunes and Oliveira 2003] is another algorithm based on PrefixSpan [Pei et al. 2001] that considers gap-constraints. More recent works on gapped sequence mining appear in [Li and Wang 2008; Zhu and Wu 2007]. Regular expressions and other constraints have been studied in [Garofalakis et al. 2002; Zhang et al. 2005]. An approach in which even large gaps between frequent patterns can be bridged was proposed in [Szymanski and Zhang 2004] in the so-called Recursive Data Mining (RDM) approach. The system was used initially for intrusion detection and more recently for text mining and role detection [Chaoji et al. 2008]. In contrast to all of these approaches, we explicitly mine the *variable gap* sequences while keeping track of all the intermediate symbols (in the gaps), their frequencies, and the gap frequency distributions. This analysis of the gap is used, later on, for building the VOGUE state model.

## 2.2 Markov Models and HMMs

Markov models and HMMs have been proposed to model longer range dependencies. However, such models suffer from high state-space complexity, since a  $k$ -th order Markov model/HMM, with alphabet  $\Sigma$ , can potentially have  $|\Sigma|^k$  states. Estimating the joint probabilities of each  $k$ -th order state is also difficult. Below we briefly review the state-of-the-art approaches for these methods.

**2.2.1 Markov Models.** The all- $k$ -order Markov model was proposed in [Pitkow and Pirolli 1999], 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 based on frequency, transition probability, and error thresholds, were proposed in [Deshpande and Karypis 2001] to improve the prediction accuracy and coverage, and to lower the state complexity of the all  $k$ -order Markov model. Another approach, called  $WM_o$ , combines the mining of sequences with a Markov predictor for web prefetching [Nanopoulos et al. 2003]. It also maintains various Markov models up to length  $k$ , but was shown to be a generalization of the above approaches. The main limitations of all the above methods is that multiple models still have to be maintained, and these methods do not explicitly model gaps.

In [Saul and Jordan 1999], mixed order Markov models were proposed. However, they rely on Expectation Maximization (EM) algorithms that are prone to local optima. Furthermore, their approach depends on a mixture of bigrams over  $k$  consecutive previous states, whereas VOGUE automatically ignores irrelevant states. Probabilistic Suffix Automata and Prediction Suffix Trees (PST), proposed in [Ron et al. 1996], are both variable memory Markov models. To learn the PST, their algorithm grows the suffix tree starting from a root node. New nodes are added to the tree if the sample supports the string labeled with that node. The Variable Length Markov Chains (VLMC) approach [Buhlmann and Wyner 1999] also uses a tree context, and focuses on minimizing the number of states via an information-theoretic approach. Unlike VOGUE, neither PST nor VLMC explicitly handle gaps, and in any case they are not hidden.

**2.2.2 HMMs.** One of the first approaches was to extend HMMs to second-order [Kriouile et al. 1990]. They extended the Viterbi and Baum-Welch algorithms [Durbin et al. 1998] that are used for state prediction and training, to directly handle second-order models. Higher-order HMMs for a fixed length  $k \geq 2$ , were considered in [du Preez 1998]. The approach first converts the fixed-order HMM to an equivalent first-order HMM, and then uses an incremental method to train the higher-order HMM from lower order models. Another fixed-order HMM was proposed in [Law and C.Chan 1996] in the context of language modeling via  $n$ -grams (for a fixed  $n$ ). Unlike the fixed-order approaches, VOGUE is variable order, and it uses a more effective and efficient approach of frequent sequence mining to extract the dependencies. Mixed order HMMs were proposed in [Schwardt and Du Preez 2000], where a prediction suffix tree is incorporated in the training of the HMM. The use of the suffix tree implies that only consecutive subsequences are considered in the modeling, and further, the method still relies on an EM method for training. In contrast, VOGUE is better able to capture the variable dependencies, with gaps, via sequence mining.

In more recent work, the Episode Generating HMMs (EGH) [Laxman et al. 2005] are especially relevant. Their main goal was to develop a generative model for each frequent episode (or sequence) in the data. To achieve this aim, they first mine frequent sequences from the data. However, they mine only non-overlapping sequences, and further they do not explicitly handle gaps as we do in VOGUE. In the second step, they build a specific HMM for a specific sequence, i.e., there are as many EGHs as there are mined sequences. In contrast to this, VOGUE is a *single* variable-order HMM incorporating all the frequent sequences. The variable-length HMM approach (VLHMM) [Wang et al. 2006] is built upon variable memory Markov chains [Ron et al. 1996; Buhlmann and Wyner 1999]; it uses a prediction prefix tree to store the context, and relies on an EM method for training. Moreover, it requires the number of states as input from the user. It does not use any form of pattern mining, as in VOGUE, to learn the model. Another recent approach combines motif discovery with HMM learning (called HPHMM) [Galassi et al. 2007]. They use a profile HMM [Eddy 1998] to learn the motifs (via local string alignment), and then combine them into a (two-level) Hierarchical HMM [Fine et al. 1998]. Unlike the general purpose variable-order HMM architecture used in VOGUE, HPHMM uses the restricted class of left-to-right models. Their motif discovery method is based on a pairwise comparison among all input sequences, whereas VOGUE directly mines frequent sequences over the entire set of input sequences. Finally, HPHMM does model the gap length via a separate HMM, but it does not model the gap symbols, i.e., it emits a random symbol in a gap state. In contrast, VOGUE simultaneously models all non-consecutive patterns, as well as gap symbols and duration statistics.

### 3. VOGUE: VARIABLE ORDER AND GAPPED HMM

As noted above, 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  $\Sigma = \{A, \dots, K\}$  and the sequence be  $S = \mathbf{ABACBDAEFBGHAIJKB}$ . We can observe that  $AB$  is a subsequence that repeats frequently (4 times), but with variable length gaps in-between.  $BA$  is also frequent (3 times), again with gaps of variable lengths. A first-order Markov model will fail to capture any patterns since no symbol depends purely on the previous symbol; a first order HMM will also have trouble modeling this sequence. We could try higher order Markov models and HMMs, but they will model many irrelevant parts of the input sequence. More importantly, *no fixed-order model for  $k \geq 1$  can learn this sequence*, since none of them will detect 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  $AB$ , and  $BA$  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 lengths of the gaps and the intervening symbols.

The training and testing of VOGUE consists of three main steps: **(i) pattern mining** via a novel Variable-Gap Sequence (VGS) mining algorithm, **(ii) data modeling** using a novel Variable-Order HMM, and **(iii) interpretation** of new data via a modified Viterbi method [Durbin et al. 1998], called Variable-Gap Viterbi (VG-Viterbi), to model the most probable path through a VOGUE model. Details of these steps appear below.

### 3.1 Mining Variable Gap Sequences

Let  $\Sigma = \{v_1, v_2, \dots, v_m\}$  be the alphabet of  $m$  distinct symbols. A *sequence* is an ordered list of symbols, and is written as  $S = v_1 v_2 \dots v_q$ , where  $v_i \in \Sigma$ . A sequence of length  $k$  is also called a *k-sequence*. For example,  $BAC$  is a 3-sequence. A sequence  $S = s_1 \dots s_n$  is a *subsequence* of another sequence  $R = r_1 \dots r_m$ , denoted as  $S \subseteq R$ , if there exist integers  $i_1 < i_2 < \dots < i_n$  such that  $s_j = r_{i_j}$  for all  $s_j$ . For example the sequence  $BAD$  is a subsequence of  $ABEACD$ . If  $S \subseteq R$ , we also say that  $R$  *contains*  $S$ .

Given a database  $D$  of sequences and a sequence  $S = s_1 \dots s_n$ , the absolute *support* or *frequency* of  $S$  in  $D$  is defined as the total number of occurrences of  $S$  across all sequences in  $D$ . Note that this definition allows for multiple occurrences of  $S$  in the same sequence. Given a user-specified threshold called the *minimum support* (denoted *minsup*), we say that a sequence is *frequent* if it occurs at least *minsup* times. We use the maximum gap threshold *maxgap* to limit the maximum gap allowed between any two elements of a  $k$ -sequence. We use the notation  $F_k$  to denote the set of all frequent  $k$ -sequences satisfying the *maxgap* constraint between any two elements.

#### 3.1.1 Variable-Gap Sequence Mining Algorithm (VGS).

VGS is based on cSPADE[Zaki 2000; 2001], 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 [Antunes and Oliveira 2003; Zaki 2000], 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 then used by 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. An example is shown in Table I. Let  $S = ACBDAHCBADFGAIEB$  be an input sequence over the alphabet  $\Sigma = \{A, \dots, I\}$ , and let *maxgap* = 2, *minsup* = 2 and

$$S = ACBDAHCBADFGAIEB$$

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

subsequence	freq	$g = 0$	$g = 1$	$g = 2$	gap symbols
<i>AB</i>	3	0	1	2	C(2), E, H, I
<i>AC</i>	2	1	1	0	H
<i>AD</i>	2	1	0	1	B, C
<i>BA</i>	2	1	1	0	D
<i>BD</i>	2	1	1	0	A
<i>CA</i>	2	0	1	1	B(2), D
<i>CB</i>	2	2	0	0	-
<i>CD</i>	2	0	1	1	A, B(2)
<i>DA</i>	2	1	0	1	F, G

Table I. VGS: Subsequences of Length 1 and 2, and Gap Information

$k = 2$ . VGS first mines the frequent subsequences of length 1, as shown. Those symbols that are frequent are extended to consider sequences of length 2. For example, *AB* 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 *AB* 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 *AB*, 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.

VGS uses the vertical database format [Zaki 2001], where an id-list is kept for each item in the dataset. Each entry in the id-list is a  $(sid, eid)$  pair, where *sid* identifies the sequence id in the data set and *eid* defines the position in this sequence at which the item appears. With the vertical idlists, computing the frequent items  $F_1$  is straightforward. We simply note all the occurrences of an item within each sequence as well as across sequences. We next test for the frequency of the candidate 2-sequences by joining the id-lists of pairs of items. A new occurrence is noted if the difference between the *eid* of the two items is less than the allowed *maxgap*. For each frequent 2-sequence, we then count the occurrences for each value of the gap length  $g$  between its elements, with  $g \in \{0, \dots, maxgap\}$ , where  $g = 0$  means that there is no gap. Longer sequences are obtained via a depth-first search and id-list joins as in SPADE [Zaki 2001], with the key difference being that we record the gaps between elements of the sequence. For example, for a 3-sequence *XYZ*, we have to note the values of the gaps between both *XY* and *YZ*. In this manner VGS computes all frequent sequences up to a maximum length  $k$  with variable gaps between elements.

### 3.2 The Basic VOGUE Model

VOGUE uses the mined sequences to build a variable order/gap HMM. 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. For clarity of exposition, we will first illustrate the working of VOGUE using mined sequences of length  $k = 2$ , and later we will extend the model for any value of  $k \geq 2$ . 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  $S : v_1 v_2$ , where  $v_1, v_2 \in \Sigma$ . Let  $\Gamma = \{v_1 | v_1 v_2 \in F_2\}$  be the set of all the distinct symbols in the first positions, and  $\Theta = \{v_2 | v_1 v_2 \in F_2\}$  be the set of all the distinct symbols in the second positions of all sequences in  $F_2$ . The basic VOGUE model is specified by the 6-tuple  $\lambda = \{\Sigma, Q, B, A, \Delta, \pi\}$  where each component is defined below.

3.2.1 *Alphabet ( $\Sigma$ )*. The alphabet for VOGUE is given as:

$$\Sigma = \{v_1, \dots, v_M\}$$

where  $|\Sigma| = M$  is the alphabet size. The alphabet must be specified upfront, or alternately VOGUE simply uses the set 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 I. For the example sequence  $S$  shown in the table, we have nine distinct symbols, thus  $M = 9$  (note that in this case, we retain all symbols, not just the frequent ones).

3.2.2 *Set of States ( $Q$ )*. The set of states in VOGUE is given as:

$$Q = \{q_1, \dots, q_N\},$$

where:

$$|Q| = N = N_1 + G + N_2 + 1$$

Here,  $N_1 = |\Gamma|$  and  $N_2 = |\Theta|$  are the number of distinct symbols in the first and second positions, respectively. Each frequent sequence  $v_a v_b \in F_2$  having a gap  $g \geq 1$  requires a gap state to model the gap between  $v_a$  and  $v_b$ .  $G$  thus gives the number of gap states required. Finally the 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:

$$Q = Q_1^s \cup Q^g \cup Q_2^s \cup Q^u$$

where the first  $N_1$  states belong to  $Q_1^s$ , the next  $G$  states belong to  $Q^g$ , the following  $N_2$  states belong to  $Q_2^s$  and the universal gap state belongs to  $Q^u$ . We call members of  $Q_i^s$  the symbol states ( $i = 1, 2$ ), and members of  $Q^g$ , the gap states.

For our example  $S$  from Table I we have  $N_1 = 4$ , since there are four distinct starting symbols in  $F_2$  (namely,  $A, B, C, D$ ). We also have four ending symbols, giving  $N_2 = 4$ . The number of gap states is the number of sequences of length 2 with at least one occurrence with gap  $g \geq 1$ . Thus  $G = 8$ , since  $CB$  is the only sequence that has all consecutive ( $g = 0$ ) occurrences. With one universal gap state our model yields  $N = 4 + 8 + 4 + 1 = 17$  states. As shown in the VOGUE HMM in Figure 1, we have  $Q_1^s = \{q_1, \dots, q_4\}$ ,  $Q_2^s = \{q_{13}, \dots, q_{16}\}$ ,  $Q^g = \{q_5, \dots, q_{12}\}$ , and  $Q^u = \{q_{17}\}$ .

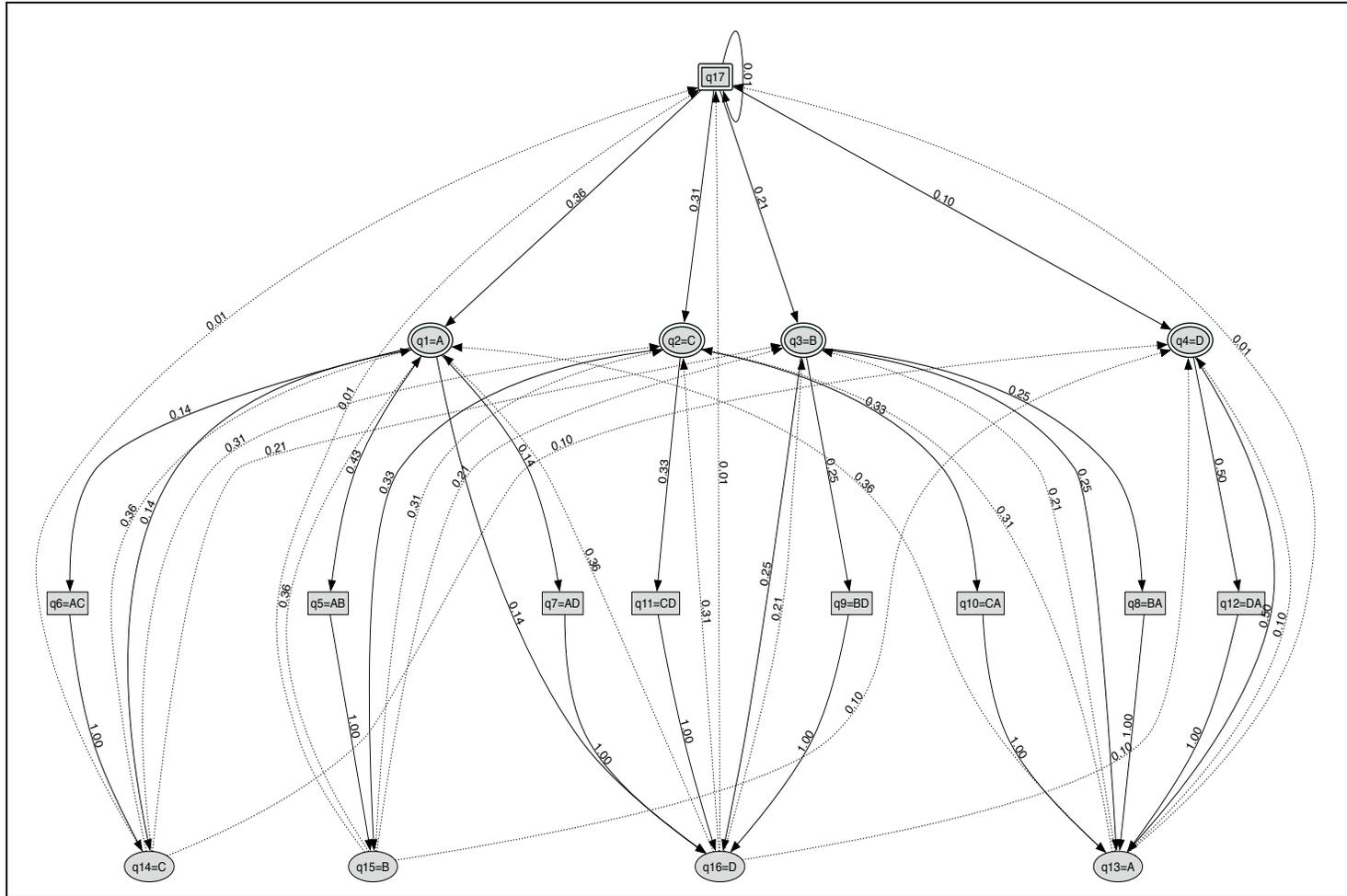


Fig. 1. Variable-order VOGUE HMM: The first symbol states  $Q_1^s$  are shown as double ellipse shaped, the second symbol states  $Q_2^s$  are shown as ellipse shaped, the gap states  $Q^g$  are shown as box shaped, and the universal gap  $Q^u = \{q_N\}$  is shown as double box shaped.

3.2.3 *Symbol Emission Probabilities (B)*. The emission probability matrix is 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  $P(v_m|q_i)$  is the probability of emitting symbol  $v_m$  in state  $q_i$ . This probability differs depending on whether  $q_i$  is a gap state or not. We assume that each first symbol state  $q_i \in Q_1^s$  outputs only a unique symbol. Likewise, each second symbol state in  $Q_2^s$  also emits only a unique symbol. We will say that a symbol state  $q_i \in Q_1^s$  or  $q_i \in Q_2^s$ , emits the unique symbol  $v_i \in \Sigma$ . The gap states ( $q_i \in Q^g \cup Q^u$ ) may output different symbols depending on the symbols observed in those gaps. Note that since there is one gap state for each  $v_a v_b \in F_2$ , we will use the notation  $q_{ab}$  to denote the gap state between  $v_a$  and  $v_b$ . 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 a very small probability of emission of such a symbol in the gap states.

3.2.3.1 *Symbol States*. The emission probability for symbol states,  $q_i \in Q_1^s$  or  $q_i \in Q_2^s$ , is given as follows:

$$b(q_i, v_m) = \begin{cases} 1, & \text{if } v_m = v_i \\ 0, & \text{otherwise} \end{cases}$$

3.2.3.2 *Gap States*. If  $q_{ab}$  is in  $Q^g$ , its emission probability depends on the symbol distribution established by VGS. Let  $\Sigma_{ab}$  be the set of symbols that were observed by VGS in the gap between  $v_a$  and  $v_b$  for any  $v_a v_b \in F_2$ . Let  $freq_{ab}(v_m)$  denote the frequency of observing the symbol  $v_m$  between  $v_a$  and  $v_b$ . For the gap state  $q_{ab}$  the emission probabilities are given as:

$$b(q_{ab}, v_m) = \left( \frac{freq_{ab}(v_m)}{\sum_{v_j \in \Sigma_{ab}} freq_{ab}(v_j)} \right) \times \sigma + \frac{1}{M} \times (1 - \sigma)$$

where  $\sigma$  acts as a smoothing parameter to handle the case when  $v_m$  is a previously unseen symbol in the training phase. For instance we typically set  $\sigma = 0.99$ , so that the second term ( $\frac{1-\sigma}{M} = \frac{0.01}{M}$ ) essentially serves as a pseudo-count to make the probability of unseen symbols non-zero.

3.2.3.3 *Universal Gap*. For the universal gap  $q_N$  we have:

$$b(q_N, v_m) = \left( \frac{freq(v_m)}{\sum_{v_m \in \Sigma} freq(v_m)} \right) \times \sigma + \frac{1}{M} \times (1 - \sigma)$$

where  $freq(v_m)$  is simply the frequency of symbol  $v_m$  established by VGS. In other words,  $v_m$  is emitted with probability proportional to its frequency in the training data.  $\sigma$  (set to 0.99) is the smoothing parameter as described above.

In our running example from Table I, for the symbol  $v_m = C$  and the gap state  $q_5$  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. Thus  $C$ 's emission probability is  $\frac{2}{5} \times 0.99 + \frac{0.01}{9} = 0.397$ . The complete set of symbol emission probabilities for topology in Figure 1 is shown in Table II.

		A	B	C	D	E	F	G	H	I
First Symbol States $Q_1^s$	$q_1$	1.000								
	$q_2$			1.000						
	$q_3$		1.000							
	$q_4$				1.000					
Gap States $Q^g$	$q_5$	0.001	0.001	0.397	0.001	0.199	0.001	0.001	0.199	0.199
	$q_6$	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.991	0.001
	$q_7$	0.001	0.496	0.496	0.001	0.001	0.001	0.001	0.001	0.001
	$q_8$	0.001	0.001	0.001	0.991	0.001	0.001	0.001	0.001	0.001
	$q_9$	0.991	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
	$q_{10}$	0.001	0.661	0.001	0.331	0.001	0.001	0.001	0.001	0.001
	$q_{11}$	0.331	0.661	0.001	0.001	0.001	0.001	0.001	0.001	0.001
	$q_{12}$	0.001	0.001	0.001	0.001	0.001	0.496	0.496	0.001	0.001
Second Symbol States $Q_2^s$	$q_{13}$	1.000								
	$q_{14}$			1.000						
	$q_{15}$		1.000							
	$q_{16}$				1.000					
$Q^u$	$q_{17}$	0.249	0.187	0.125	0.125	0.063	0.063	0.063	0.063	0.063

Table II. Symbol Emission Probability Matrix ( $B$ )

3.2.4 *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. A transition of a second symbol state can go back to either one of the first symbol states or to the universal gap state. Finally, a transition from the universal gap state can go to any of the starting states or the intermediate gap states. We discuss these cases below.

3.2.4.1 *Transitions from First States*. Any first symbol state  $q_i \in Q_1^s$  may transition only to either a second symbol state  $q_j \in Q_2^s$  (modeling a gap of  $g = 0$ ) or to a gap state  $q_j \in Q^g$  (modeling a gap of  $g \in [1, maxgap]$ ). Note that the symbol states  $Q_1^s$  and  $Q_2^s$  only emit one distinct symbol as described above, so we can associate each state  $q_i$  with the corresponding symbol it emits, say  $v_i \in \Sigma$ . Let  $v_a v_b \in F_2$  be a frequent 2-sequence uncovered by VGS. Let  $freq(v_a, v_b)$  denote the total frequency of the sequence, and let  $freq(v_a, v_b, g)$  denote the number of occurrences of  $v_a$  followed by  $v_b$  after a gap length of  $g$ , with  $g \in [1, maxgap]$ , i.e.,  $freq(v_a, v_b) = \sum_{g=0}^{maxgap} freq(v_a, v_b, g)$ . The transition probabilities from  $q_i \in Q_1^s$  are

then given as:

$$a(q_i, q_j) = \begin{cases} \frac{freq(v_i, v_j, 0)}{\sum_{v_i, v_b \in F_2} freq(v_i, v_b)}, & \text{if } q_j \in Q_2^s \\ \frac{\sum_{g>0} freq(v_i, v_j, g)}{\sum_{v_i, v_b \in F_2} freq(v_i, v_b)}, & \text{if } q_j \in Q^g \\ 0, & \text{otherwise} \end{cases}$$

3.2.4.2 *Transitions from Gap States.* Any gap state  $q_{ab} \in Q^g$  may transition only to the second symbol state  $q_b \in Q_2^s$ . For  $q_i = q_{ab} \in Q^g$  we have:

$$a(q_i, q_j) = \begin{cases} 1, & \text{if } q_j = q_b \in Q_2^s \\ 0, & \text{otherwise} \end{cases}$$

3.2.4.3 *Transitions from Second States.* A second symbol state  $q_i \in Q_2^s$  may transition only to either first symbol state  $q_j \in Q_1^s$  (modeling a gap of  $g = 0$ ), or to the universal gap state  $q_N$  (modeling other gaps). For  $q_i \in Q_2^s$  we thus have:

$$a(q_i, q_j) = \begin{cases} \sigma \times \frac{\sum_{q_b \in Q_2^s} freq(v_j, v_b)}{\sum_{v_a, v_b \in F_2} freq(v_a, v_b)}, & \text{if } q_j \in Q_1^s \\ 1 - \sigma, & \text{if } q_j = q_N \in Q^u \\ 0, & \text{otherwise} \end{cases}$$

Here  $\sigma = 0.99$  acts as the smoothing parameter, but this time for state transitions, so that there is a small probability ( $1 - \sigma = 1 - 0.99 = 0.01$ ) of transitioning to  $Q^u$ . Transitions back to first states are independent of second symbol state  $q_i$ . 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 mined by VGS. We assign an empirically chosen value of 1% to the transition from the second states  $Q_2^s$  to the universal gap state  $q_N$ . Furthermore, to satisfy  $\sum_{j=1}^N a(q_i, q_j) = 1$ , we assign the remaining 99% to the transition to the first states  $Q_1^s$ .

3.2.4.4 *Transitions from Universal Gap.* The universal gap state can only transition to the first states or to itself. For  $q_i = q_N$  we have:

$$a(q_i, q_j) = \begin{cases} \sigma \times \frac{\sum_{q_b \in Q_2^s} freq(v_j, v_b)}{\sum_{v_a, v_b \in F_2} freq(v_a, v_b)}, & \text{if } q_j \in Q_1^s \\ 1 - \sigma, & \text{if } q_j = q_N \\ 0, & \text{otherwise} \end{cases}$$

Here  $\sigma = 0.99$  weighs the probability of transitioning to a first symbol state, whereas  $1 - \sigma = 0.01$  weighs the probability of self transition. These values were chosen empirically. Table III shows transitions between states and their probabilities in VOGUE for our running example in Table I (see Figure 1 for the model topology).

3.2.5 *State Duration Probabilities ( $\Delta$ ).* Each state's duration is considered explicitly within that state. Here, we treat the duration as the number of symbols to emit from that state. The probability of generating a given number of symbols is given by the state duration probability matrix:

$$\Delta = \{\Delta(q_i, d) | d \in [1, maxgap]\},$$

where  $d$  gives the duration, which ranges from 1 to *maxgap*.

	First Symbol States ( $Q_1^s$ )				Gap States ( $Q^g$ )								Second Symbol States ( $Q_2^s$ )				$Q^u$
	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$	$q_7$	$q_8$	$q_9$	$q_{10}$	$q_{11}$	$q_{12}$	$q_{13}$	$q_{14}$	$q_{15}$	$q_{16}$	
$q_1$					0.43	0.14	0.14							0.14		0.14	
$q_2$										0.33	0.33				0.33		
$q_3$							0.25	0.25					0.25			0.25	
$q_4$												0.50	0.50				
$q_5$														1.00	1.00		
$q_6$														1.00		1.00	
$q_7$																1.00	
$q_8$													1.00				1.00
$q_9$																	1.00
$q_{10}$													1.00				
$q_{11}$																	1.00
$q_{12}$													1.00				
$q_{13}$	0.36	0.31	0.21	0.10													0.01
$q_{14}$	0.36	0.31	0.21	0.10													0.01
$q_{15}$	0.36	0.31	0.21	0.10													0.01
$q_{16}$	0.36	0.31	0.21	0.10													0.01
$q_{17}$	0.36	0.31	0.21	0.10													0.01

Table III. State Transition Probability Matrix ( $A$ )

For a symbol state,  $q_i \in Q_j^s$ , the duration is always 1, i.e., we always emit one symbol. For gap states, let  $q_i = q_{ab}$  be the gap state between the state  $q_a \in Q_1^s$  and the state  $q_b \in Q_2^s$  corresponding to the sequence  $v_a v_b \in F_2$ . The duration probability for  $q_i$  will then be proportional to the frequency of observing a given gap length value for that sequence. Putting the two cases together, we have

$$\Delta(q_i, d) = \begin{cases} \frac{\text{freq}(v_a, v_b, d)}{\sum_{g>0} \text{freq}(v_a, v_b, g)}, & q_i = q_{ab} \in Q^g \\ 1, & q_i \in Q_1^s \cup Q_2^s \text{ and } d = 1 \\ 0, & \text{otherwise} \end{cases}$$

For the gap states, the idea is to model the “gap duration”, which corresponds to the number of gap symbols emitted from that state, which is proportional to the probability of that gap value for that gap state. For instance, in our running example, for the gap state  $g_4$  between the states that emit  $A$  and  $B$ , we have  $\Delta(g_4, 2) = \frac{2}{3} = 0.67$ , since we twice observe a gap of 2, out of three occurrences. The gap duration probabilities for our running example are shown in Table IV.

	$q_5$	$q_6$	$q_7$	$q_8$	$q_9$	$q_{10}$	$q_{11}$	$q_{12}$	Other $q_i$
d=1	0.33	1.0		1.0	1.0	0.5	0.5		1.0
d=2	0.67		1.0			0.5	0.5	1.0	1.0

Table IV. State Duration Probabilities ( $\Delta$ )

3.2.6 *Initial State Probabilities ( $\pi$ )*. 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:

$$\pi(i) = \begin{cases} \sigma \times \frac{\sum_{q_j \in Q_2^s} \text{freq}(v_i, v_j)}{\sum_{v_a v_b \in F_2} \text{freq}(v_a, v_b)}, & \text{if } q_i \in Q_1^s \\ 1 - \sigma, & \text{if } q_i = q_N \in Q^u \\ 0, & \text{otherwise} \end{cases}$$

We use a small value ( $1 - \sigma = 1 - 0.99 = 0.01$ ) for the universal gap state as opposed to the states in  $Q_1^s$  to accentuate the patterns retained by VGS while still providing a possibility for gaps after and before them. Note that the empirical value of  $\sigma = 0.99$  used in the transition and emission probabilities works essentially like pseudo-counts [Durbin et al. 1998] to allow for symbols that are unseen in the training data set or to allow the model to transition to less likely states.

#### 4. THE GENERAL VOGUE MODEL

We now generalize the basic VOGUE model to handle frequent  $k$ -sequences, with  $k \geq 2$ . Let  $F$  be the set of all frequent subsequences mined by VGS, and let  $k$  be the maximum length of any sequence. Let  $F(j)$  be the set of subsequences in  $F$  of length at least  $j$ , and let a  $l$ -sequence be denoted as  $v_1 v_2 \cdots v_l$ . Let  $\Gamma(j)$  be the set of symbols in the  $j^{\text{th}}$  position across all the subsequences in  $F(j)$ , then  $\Gamma(k)$  is then the set of different last symbols across all subsequences in  $F$ . The VOGUE model is specified by the 6-tuple  $\lambda = \{\Sigma, Q, A, B, \Delta, \pi\}$ , as before. These components are described below.

Let  $N_i = |\Gamma(i)|$ ,  $i = 1, \dots, k$  denote 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. Let  $G_i$  (for  $i < k$ ) be the number of distinct pairs of symbols in positions  $i$  and  $i + 1$  across all sequences in  $F(i + 1)$ . This corresponds to the number of gap states required between states at positions  $i$  and  $i + 1$ .

#### 4.1 Alphabet ( $\Sigma$ )

The alphabet  $\Sigma = \{v_1, \dots, v_M\}$  is the number of different symbols seen in the training set.

#### 4.2 Set of States ( $Q$ )

For the general VOGUE model, the set of states is given as  $Q = \{q_1, \dots, q_N\}$ , where  $N + 1 = N_1 + G_1 + \dots + N_{i-1} + G_{i-1} + \dots + N_k + 1$ . Here  $q_N$  is the universal gap state, as before. We assume further that the states are partitioned into the “symbol” and “gap” states in the given order. That is, the first  $N_1$  states are the first symbol states, the next  $G_1$  states are the gap states between the first and second symbol states, the next  $N_2$  states are the second symbol states and so on. Let  $Q_i^s$  denote the set of  $i$ -th symbol states, i.e., those at position  $i$ , given by  $\Gamma(i)$ , with  $N_i = |\Gamma(i)|$ . Let  $Q_i^g$  denote the  $i$ -th gap states, i.e., those between the  $i$ -th and  $(i + 1)$ -th symbol states.

#### 4.3 Symbol Emission Matrix ( $B$ )

A symbol state  $q_i \in Q_i^s$ , can emit only one symbol  $v_i \in \Gamma(j)$ . The emission probability from  $q_i$  is then given as:

$$b(q_i, v_m) = \begin{cases} 1, & \text{if } v_m = v_i \\ 0, & \text{otherwise} \end{cases}$$

Let the gap state  $q_i \in Q_j^g$  be the gap state between the states  $q_a \in Q_j^s$  and  $q_b \in Q_{j+1}^s$ , which we denote as  $q_i = q_{ab}$ . Across all the sequences in  $F(j + 1)$ , let  $freq_{ab}(v_m, j, j + 1)$  denote the total frequency of observing the symbol  $v_m$  between  $v_a$  and  $v_b$  at positions  $j$  and  $j + 1$  respectively. The emission probabilities from  $q_i \in Q_j^g$ , are given as:

$$b(q_{ab}, v_m) = \left( \frac{freq_{ab}(v_m, j, j + 1)}{\sum_{v_c} freq_{ab}(v_c, j, j + 1)} \right) \times \sigma + \frac{1 - \sigma}{M}$$

For  $q_i = q_N$ , the universal gap state, we have:

$$b(q_N, v_m) = \left( \frac{freq(v_m)}{\sum_{v_m \in \Sigma} freq(v_m)} \right) \times \sigma + \frac{1 - \sigma}{M}$$

where  $freq(v_m)$  is simply the frequency of symbol  $v_m$  as mined by VGS.

#### 4.4 Transition Matrix ( $A$ )

There are four main cases to consider for the transitions between states. The symbol states in  $Q_i^s$  may transition to (i) gap states  $Q_i^g$ , or (ii) directly to the symbol states at position  $i + 1$ , namely  $Q_{i+1}^s$  for  $i \in [1, k - 1]$ , or (iii) to the universal gap state  $q_N$

(to start a fresh cycle through the model). A gap state in  $Q_i^g$  may only transition to the symbol state  $Q_{i+1}^s$ , for  $1 \leq i < k$ . The universal gap state may transition to any of the first symbol states  $Q_1^s$  or to itself.

Let  $q_i \in Q_p^s$  be a  $p$ -th symbol state. Let  $freq_p(v_i)$  denote the frequency of observing the symbol  $v_i$  at position  $p$  across all sequences in  $F(p)$ . Let  $freq_p(v_i, v_j)$  denote the total frequency of observing the symbol  $v_i$  at position  $p$  and  $v_j$  at position  $p+1$  across all sequences in  $F(p+1)$ . Further, let  $freq_p(v_i, v_j, g)$  denote the frequency for a given gap value  $g$ . Then, transition probabilities for a symbol state  $q_i \in Q_p^s$  are:

$$a(q_i, q_j) = \begin{cases} \frac{freq_p(v_i, v_j, 0)}{freq_p(v_i)}, & \text{if } q_j \in Q_{p+1}^s \\ \frac{\sum_{g>0} freq_p(v_i, v_j, g)}{freq_p(v_i)}, & \text{if } q_j \in Q_p^g \\ 1 - \frac{freq_p(v_i, v_j)}{freq_p(v_i)}, & \text{if } q_j = q_N \\ 0, & \text{otherwise} \end{cases}$$

A gap state  $q_i = q_{ab} \in Q_p^g$ , may transition only to a symbol state  $q_j \in Q_{p+1}^g$  at position  $p+1$ :

$$a(q_i, q_j) = \begin{cases} 1, & \text{if } q_j = q_b \in Q_{p+1}^g \\ 0, & \text{otherwise} \end{cases}$$

Finally, the universal gap state  $q_N$  may transition to itself or to any of the first states  $Q_1^s$ :

$$a(q_N, q_j) = \begin{cases} \sigma \times \frac{freq_1(v_j)}{\sum_{v_b \in \Sigma} freq_1(v_b)}, & \text{if } q_j \in Q_1^s \\ 1 - \sigma, & \text{if } q_j = q_N \\ 0, & \text{otherwise} \end{cases}$$

As before we set  $\sigma = 0.99$ .

#### 4.5 State Duration Probabilities ( $\Delta$ )

As before, for symbol states the duration is always  $d = 1$ . For the gap state  $q_i \in Q_p^g$ , assuming that  $q_i = q_{ab}$ , i.e., it lies between the symbol  $v_a \in Q_p^s$  and  $v_b \in Q_{p+1}^s$ , we have:

$$\Delta(q_i, d) = \begin{cases} \frac{freq(v_a, v_b, d)}{\sum_{g>0} freq(v_a, v_b, g)}, & q_i = q_{ab} \in Q_p^g \\ 1, & q_i \in Q_j^s \text{ and } d = 1 \\ 0, & \text{otherwise} \end{cases}$$

### 5. DECODING: FINDING THE BEST STATE SEQUENCE

After extracting the patterns and modeling the data, the model is ready to be used to find for the given newly observed sequence the best matching state sequence in the model. This problem is referred to in HMMs [Rabiner 1989] as the *decoding* problem. This problem is difficult to solve even though there are several possible ways of solving it. One possible solution is to choose individually the most likely states for each position in the test sequence. However, while this approach maximize the states that will explain the observation sequence, it may yield an infeasible sequence of the states, for instance, if some of the transitions needed to generate the model sequence have zero probabilities for the corresponding model states. The

best matching state sequence can be found using the dynamic programming based Viterbi algorithm [Durbin et al. 1998].

Due to VOGUE's unique structure, we modified the Viterbi algorithm to handle the notion of duration in the states. We call this new algorithm Variable-Gap Viterbi (VG-Viterbi). Given a test sequence  $O = o_1 o_2 \cdots o_T$ , VG-Viterbi finds the sequence of states from the model  $\lambda$ , as well as the state durations, whose traversal generate the sequence that best matches the observed sequence. That is, VG-Viterbi finds a sequence of states  $q^* = \{q_1^*, q_2^*, \dots, q_T^*\}$  from the model  $\lambda$  such that:

$$q^* = \arg \max_q P(O, q | \lambda)$$

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 position  $t$ :

$$\delta_t(j) = \max_{q_1 \cdots q_{t-1}} P(o_1, \dots, o_t, q_1, \dots, q_{t-1}, q_t = q_j | \lambda)$$

Assuming that each state has a corresponding duration, we can rewrite the above expression as:

$$\begin{aligned} \delta_t(j) = \max_{d_1, \dots, d_r} P( & o_1, \dots, o_{d_1}, q_1 = \dots = q_{d_1}, \\ & o_{d_1+1}, \dots, o_{d_1+d_2}, q_{d_1+1} = \dots = q_{d_1+d_2}, \\ & \vdots \\ & o_{d_1+\dots+d_{r-1}+1}, \dots, o_{d_1+\dots+d_r} = o_t, \\ & q_{d_1+\dots+d_{r-1}+1} = \dots = q_{d_1+\dots+d_r} = q_t = q_j | \lambda) \end{aligned}$$

In other words, we are assuming that the first  $d_1$  symbols  $o_1, \dots, o_{d_1}$  are being emitted from a single state, say  $s_1$ , with a duration of  $d_1$ . The next  $d_2$  symbols are emitted from state  $s_2$  with a duration of  $d_2$ , and so on. Finally, the last  $d_r$  symbols are emitted from state  $s_r$  with duration  $d_r$ . The probability is taken to be the maximum over all such values of state durations,  $d_1, \dots, d_r$ . We can rewrite the above expression purely in terms of the states  $s_i$  and their durations  $d_i$ , as follows:

$$\delta_t(j) = \max_{s_1 \cdots s_r, d_1, \dots, d_r} P(o_1, \dots, o_t, s_1, \dots, s_{r-1}, s_r = q_j | \lambda)$$

Separating out the state  $s_{r-1}$ , which we assume is the state  $q_i$ , with  $i \neq j$ , we get:

$$\delta_t(j) = \max_{\substack{s_1 \cdots s_r, \\ d_1, \dots, d_r}} P(o_1, \dots, o_t, s_1, \dots, s_{r-2}, s_r = q_j | s_{r-1} = q_i, \lambda) \cdot P(s_{r-1} = q_i | \lambda)$$

Further separating out the symbols from the last state  $s_r$ , we get:

$$\begin{aligned} \delta_t(j) = \max_{\substack{s_1 \cdots s_r, \\ d_1, \dots, d_r}} P(o_1, \dots, o_{t-d_r}, s_1, \dots, s_{r-2} | & o_{t-d_r+1}, \dots, o_t, s_{r-1} = q_i, s_r = q_j, \lambda) \cdot \\ & P(o_{t-d_r+1}, \dots, o_t | s_{r-1} = q_i, s_r = q_j, \lambda) \cdot \\ & P(s_r = q_j | s_{r-1} = q_i, \lambda) \cdot \\ & P(s_{r-1} = q_i | \lambda) \end{aligned}$$

Using the Markovian assumption that each state depends only on the previous

state, we can finally rewrite the above equation as:

$$\delta_t(j) = \max_{\substack{s_1, \dots, s_r, \\ d_1, \dots, d_r}} P(o_1, \dots, o_{t-d_r}, s_1, \dots, s_{r-2}, s_{r-1} = q_i \mid o_{t-d_r+1}, \dots, o_t, s_r = q_j, \lambda) \cdot \\ P(o_{t-d_r+1}, \dots, o_t \mid s_{r-1} = q_i, s_r = q_j, \lambda) \cdot \\ P(s_r = q_j \mid s_{r-1} = q_i, \lambda)$$

Assuming that the duration distribution of a state is independent of the observations of that state, and writing  $d_r$  as  $d$ , we get the following recursive relationship:

$$\delta_t(j) = \max_{\substack{d < \min(t, \text{maxgap}) \\ i \neq j}} \delta_{t-d}(i) \cdot a(q_i, q_j) \cdot \Delta(q_j, d) \cdot \prod_{s=t-d+1}^t b(q_j, o_s) \quad (1)$$

In other words, the probability of being in state  $q_j$  at position  $t$ , is given as the product of the probability of being in state  $q_i$  at position  $t-d$ , transitioning from  $q_i$  to  $q_j$ , probability of the duration  $d$  in state  $q_j$ , and emitting the  $d$  symbols  $o_{t-d+1}, \dots, o_t$  in state  $q_j$ . The maximum is taken over all values of duration  $d$ . Using Equation 1 VG-Viterbi computes the probability of observing the sequence  $O$  given the model  $\lambda$ , by computing  $\delta_T(j)$  over all states  $q_j$  in a recursive manner. At each step it also remembers the state that gives the maximum value to obtain the final sequence of states  $q^*$ .

Given a model with  $N$  states, and a sequence  $O$  of length  $T$ , the Viterbi algorithm takes  $O(N^2T)$  [Durbin et al. 1998] time to find the optimal state sequence. This is obviously expensive when the number of states is large and the observation sequence is very long. For VG-Viterbi, we have the extra search over the state durations, which gives a complexity of  $O(N^2 \cdot \text{maxgap} \cdot T)$ . The most effective way to reduce the complexity is to exploit the inherent sparseness of the VOGUE models. For example, consider the state transition probability matrix  $A$ , given in Table III. By design, the matrix is very sparse, with allowed transitions from  $Q_1^s$  to either  $Q_2^s$  or  $Q^g$ , from  $Q^g$  to  $Q_2^s$ , from  $Q_2^s$  to  $Q_1^s$  or  $Q^u$ , and from  $Q^u$  to  $Q_1^s$  or  $Q^u$ .

Considering that many of the transitions in VOGUE are non-existent ( $a(q_i, q_j) = 0$ ), we optimized the recursion in Equation 1 to search over only the non-zero transitions. For example, for the basic VOGUE model built using  $k = 2$  length sequences, we can then obtain the complexity as follows: Given the set of frequent sequences mined by VGS, let  $F_1$  denote the frequent symbols, and let  $F_2$  denote the frequent 2-sequences. First, note that  $|Q_1^s| \leq |F_1| \ll M$  and  $|Q_2^s| \leq |F_1| \ll M$ , since the frequent symbols are generally much smaller than  $M$ , the total number of symbols in the alphabet. Second, note that  $|Q^g| = |F_2| \leq |F_1|^2 \ll M^2$ . Using the sparseness of the model to our advantage, we can reduce the number of states to search over when computing  $\delta_t(j)$ :

- If  $q_j \in Q_1^s$ , then search only over  $q_i \in Q_2^s$  or  $q_i = q_N$ .
- If  $q_j \in Q_2^s$ , then search only over  $q_i \in Q_1^s \cup Q^g$ .
- If  $q_j \in Q^g$ , then search only over  $q_i \in Q_1^s$ .
- If  $q_j = q_N$ , then search only over  $q_i \in Q_2^s$  or  $q_i = q_N$ .

Since  $|Q_1^s| \approx |Q_2^s| \approx M \ll |Q^g| \ll M^2$ , in the worst case we search over  $O(|Q^g|)$  states in the cases above. Thus the complexity of VG-Viterbi is  $O(|Q^g| \cdot N \cdot T \cdot \text{maxgap}) \ll O(N^2 \cdot T \cdot \text{maxgap})$ . The practical effect of exploiting the sparseness

is in the much reduced decoding time as opposed to a search over the full set of states at each step.

## 6. EXPERIMENTAL RESULTS AND ANALYSIS

VOGUE was implemented in Python, and all experiments were done on a 2.8Ghz quad-core Intel Xeon MacPro, with 4GB 800MHz memory, and 500GB disk, running Mac OS X 10.5.6. The VOGUE implementation and the datasets used in this paper are available as open-source at: [www.cs.rpi.edu/~zaki/software/VOGUE](http://www.cs.rpi.edu/~zaki/software/VOGUE).

### 6.1 Datasets

We tested the VOGUE model on a variety of datasets, including biological sequences, web usage logs, intrusion commands and spelling. We discuss details of these datasets below; Table V shows the number of training and testing instances in each class for the various datasets. It also shows the average sequence length per class.

PROSITE		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
Training	#Seq	34	169	64	43	97	74	113	87	16	22
	Avg. Len	814.1	311.9	371.1	684.3	250.0	252.1	120.4	959.5	341.9	511.1
Testing	#Seq	11	56	21	14	32	24	37	28	5	7
	Avg. Len	749.5	572.1	318.1	699.4	386.0	266.5	407.9	870.6	180.8	405.3

SCOP		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
Training	#Seq	6	6	6	6	6	6	6	6	6	6
	Avg. Len	182.2	153.2	103.3	52.2	57.3	78.0	68.7	276.5	177.0	125.7
Testing	#Seq	4	4	4	4	4	4	4	4	4	4
	Avg. Len	135.8	152.0	81.0	61.3	60.3	86.8	89.8	270.8	177.0	122.8

CSLOGS		edu	oth
Training	#Seq	3577	12598
	Avg. Len	5.8	6.9
Testing	#Seq	1190	4197
	Avg. Len	7.7	8.7

Intrusion		User0	User1	User2	User3	User4	User5	User6	User7	User8
Training	#Seq	423	367	568	364	685	411	1280	1006	1194
	Avg. Len	14.0	35.9	26.3	36.2	39.1	51.7	24.9	10.5	29.7
Testing	#Seq	139	121	187	120	226	135	605	333	396
	Avg. Len	13.7	46.9	8.9	21.5	40.5	91.8	19.7	11.1	37.0

Spelling		Incorrect	Correct
Training	#Seq	986	1204
	Avg. Len	24.5	27.9
Testing	#Seq	327	400
	Avg. Len	25.5	26.5

Table V. Dataset Characteristics: Number of training and testing instances per class, and the average length of the training and testing sequences.

6.1.1 *Biological Sequence Datasets: PROSITE and SCOP.* In recent years, a large amount of work in biological sequence analysis has focused on methods for finding homologous proteins [Durbin et al. 1998]. Computationally, protein sequences are treated as long strings of characters with a finite alphabet of 20 amino acids, namely,  $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$ . 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. We used two different biological datasets for our experiments: (i) the PROSITE [Falquet et al. 2002] database containing families of protein sequences, and (ii) the SCOP [Murzin et al. 1995] dataset that includes a curated classification

of known proteins structures with the secondary structure knowledge embedded in the dataset.

6.1.1.1 *PROSITE*. We used ten families from PROSITE, namely, *PDOC00662*, *PDOC00670*, *PDOC00561*, *PDOC00064*, *PDOC00154*, *PDOC00224*, *PDOC00271*, *PDOC00343*, *PDOC00397*, *PDOC00443*. We will refer to these families as  $F_1$ ,  $F_2$ ,  $\dots$ ,  $F_{10}$ , 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 amino acids. 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  $N_{train}^i$  consists of 75% of the data, while the testing data  $N_{test}^i$  contains the remaining 25%. For example,  $N_{train}^1 = 34$  and  $N_{test}^1 = 11$ . There are a total of 235 test sequences across all families. Details on the number of sequences and average sequence length for training/test sets for the different classes are given in Table V.

6.1.1.2 *SCOP*. The SCOP dataset 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. We used 10 superfamilies from the SCOP dataset, namely, family 49417, 46458, 46626, 46689, 46997, 47095, 47113, 48508, 69118, and 81296. Each family has 10 sequences. We divided each family data set into 60% (6 sequences from each family) for training and 40% for testing (4 from each family). Thus there are 40 test sequences across the 10 families.

6.1.2 *Web Usage Logs Dataset: CSLOGS*. The CSLOGS dataset consists of web logs files collected at the CS department at RPI. User sessions are expressed as subgraphs of the web graph, and contain complete history of the user clicks. Each user session has a name (IP or host name), a list of edges giving source and target pages and the time when the link was traversed. We convert the user graph into a sequence by arranging all the visited pages in increasing order of their timestamps.

The CSLOG dataset spans 3 weeks worth of such user-session sequences. To convert it into a classification dataset we chose to categorize each user-session into one of two class labels: **edu** corresponds to users from an “edu” domain, (also includes “ac” academic domain), while **oth** class corresponds to all users visiting the CS department from any other domain. The goal of classification is to find out if we can separate users who come from academic versus other domains from their browsing behavior within the CS web site at RPI.

We used the first two weeks of logs as the training, and the third week’s logs as the testing datasets, adjusted slightly so that the training set had 75%, and the testing set had 25% of the sequences. In all the CSLOGS dataset contains 16206 unique web pages, which make up the alphabet. As shown in Table V, the training dataset had 16175 sessions, with 3577 labeled as **edu** and 12598 labeled as **other**.

The testing dataset had 5387 sessions, with 1190 (22.1%) having true class `edu` and 4197 with true class `other`.

**6.1.3 Intrusion Detection Dataset.** The intrusion detection dataset deals with the problem of differentiating between masqueraders and the true user of a computer terminal. The data consists of user-specific unix command sequences. We use command stream data collected from eight UNIX users from Purdue University [Lane and Brodley 1999] over varying periods of time (USER0 and USER1 are the same person working on different projects). User data enters the system by the monitoring of UNIX shell command data [Lane and Brodley 1999], captured via the (t)csh history file mechanism. An example session could be: `ls -l; vi t1.txt; ps -eaf; vi t2.txt; ls -a /usr/bin/*; rm -i /home/*; vi t3.txt t4.txt; ps -ef;`. Each process/command in the history data together with its arguments is treated as a single token. However, to reduce the alphabet size, we omitted filenames in favor of a file count as in Lane [Lane and Brodley 1999]. For example, the user sequence given above is converted to the following set of tokens  $T = \{t_i : 0 \leq i < 8\}$ , where  $t_0 = \text{ls } -l$ ,  $t_1 = \text{vi } <1>$ ,  $t_2 = \text{ps } -eaf$ ,  $t_3 = \text{vi } <1>$ ,  $t_4 = \text{ls } -a <1>$ ,  $t_5 = \text{rm } -i <1>$ ,  $t_6 = \text{vi } <2>$ , and  $t_7 = \text{ps } -ef$ . The notation  $<n>$  gives the number of arguments ( $n$ ) of a command. For instance, the command `vi t1.txt` is tokenized as `vi <1>`, while `vi t3.txt t4.txt` as `vi <2>`.

In all there are 2354 unique commands across the users; this number thus gives the cardinality of the set of symbols (the alphabet) for the Intrusion dataset. The class-specific training (75%) and testing (25%) instances are given in Table V.

**6.1.4 Context-sensitive Spelling Correction Dataset.** We also tested our algorithm on the task of correcting spelling errors that result in valid words, such as *there* vs. *their*, *I* vs. *me*, *than* vs. *then*, and *you're* vs. *your* [Golding and Roth 1996]. For each test, we chose the two commonly confused words and searched for sentences in the 1-million-word Brown corpus [Kucera and Francis 1967] containing either word. We removed the target word and then represented each word by the word itself, the part-of-speech tag in the Brown corpus, and the position relative to the target word. For example, the sentence “And then there is politics” is translated into (word=`and` tag=`cc` pos=`-2`)  $\rightarrow$  (word=`then` tag=`rb` pos=`-1`)  $\rightarrow$  (word=`is` tag=`bez` pos=`+1`)  $\rightarrow$  (word=`politics` tag=`nn` pos=`+2`). The final dataset consists of all examples from the correct and incorrect usages, which form the two classes for the classifier. Overall 25% of the instances are used for testing, and 75% for training. The number of test and training instances in each class are shown in Table V. The alphabet size is 12,280, which is the number of unique words in the corpus.

## 6.2 Alternative HMM Models

We compare VOGUE with three different HMMs. As a baseline, we compare with a regular first-order HMM. On the biological sequences, we also compare against HMMER [Eddy 1998]. Finally, we compare with  $k^{th}$ -order HMMs on some selected datasets.

6.2.1 *First-Order HMM*. For the baseline, we used an open-source python HMM implementation called `logilab-hmm v0.5`<sup>2</sup>, which takes as input the number of states and output symbols. It initializes the state transition matrix randomly, and then uses Baum-Welch training [Rabiner 1989] to optimize the state transition and symbol emission probabilities. It uses the standard Viterbi algorithm to compute the probability of the test sequences. Baum-Welch is essentially an Expectation-Maximization algorithm which can get stuck in local minima, starting from the random initial probabilities. For a given number of states, we therefore train multiple (actually, three) models from different random starting transition matrices, and we report the best results. Furthermore, since the number of states is an input parameter, we try several values and chose the one that consistently yields better results.

6.2.2 *Higher Order HMM*. We also built several  $k$ -th order HMMs, denoted as  $k$ -HMM, for different values of  $k$ . 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 the baseline first-order HMM from above.

For the order, we considered  $k = 2$  and  $k = 4$ . Of course  $k = 1$  is the same as the baseline 1st order HMM. We could not run with higher order (with  $k > 4$ ) HMMs since the baseline HMM implementation ran into precision problems. As before, we tried different values for the number of states, and report the best results. The number of observations  $M$  for the  $k = 1$  case was set to 20 since it is the number of amino acids.  $M = 394; 17835$  were the number of observations used for  $k = 2; 4$ , respectively. These values were obtained from a count of the different new symbols used for each value of  $k$ .

6.2.3 *HMMER*. HMMER [Eddy 1998], a profile HMM, is one of the state-of-the-art approaches for biological sequence modeling and classification. To model the data using HMMER, we first need to align the training sequences using CLUSTAL-W<sup>3</sup>. We then build a profile HMM using the multiple sequence alignment and compute the scores for each test sequence using HMMER. HMMER 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.

6.2.4 *VOGUE*. We built VOGUE models with different values of *minsup* and *maxgap* but with the constant  $k = 2$  for the length of the mined sequences in VGS. We then choose the parameters that give consistently good results and use them for the comparison.

<sup>2</sup><http://www.logilab.org/project/logilab-hmm>

<sup>3</sup><http://www.ebi.ac.uk/clustalw>

### 6.3 Evaluation and Scoring

We built models for each class using each of the competing methods, namely VOGUE, HMM,  $k$ -th order HMM, and HMMER, using the training sequences for each class. That is, assuming that there are  $c$  classes in a dataset, we train  $c$  HMM models,  $M_0, M_1, \dots, M_c$ . For each test sequence, we score it against all of the  $c$  models (for each class), using the Viterbi algorithm; the predicted class is the one for which the corresponding model yields the maximum score. That is, for a test instance  $seq$ , its class is given as  $\arg \max_i \{P(seq|M_i)\}$ .

Note that we use the log-odds score for the test sequences, defined as the ratio of the probability of the sequence using a given model to the probability of the sequence using the **Null** model. That is:

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

As noted above,  $P(seq|Model)$  is computed using the Viterbi algorithm that computes the most probable path through the model. The **Null** model is a simple one state HMM that emits each observation with equal probability ( $1/|\Sigma|$ ). 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.

For comparing alternative approaches, we compute the total accuracy of the classifier, as well as class specific accuracies. We show experimentally that VOGUE's modeling power is superior to higher-order HMMs. VOGUE also outperforms regular HMMs and HMMER [Eddy 1998], a HMM model especially designed for protein sequences that takes into consideration insertions, deletions and substitutions between similar amino acids.

### 6.4 Accuracy Comparison

We now compare the classification accuracy of VOGUE with the other HMM models, on the different datasets mentioned above.

Class	VOGUE	HMMER	HMM	2-HMM	4-HMM
F1-PDOC00662	<b>81.82</b>	72.73	27.27	36.36	18.18
F2-PDOC00670	<b>80.36</b>	73.21	71.43	50.0	41.07
F3-PDOC00561	<b>95.24</b>	42.86	61.9	80.95	33.33
F4-PDOC00064	<b>85.71</b>	<b>85.71</b>	<b>85.71</b>	<b>85.71</b>	64.29
F5-PDOC00154	<b>71.88</b>	<b>71.88</b>	59.38	40.62	56.25
F6-PDOC00224	87.50	<b>100</b>	79.17	87.5	91.67
F7-PDOC00271	89.19	<b>100</b>	64.86	5.41	27.03
F8-PDOC00343	89.29	96.43	71.43	96.43	<b>100.0</b>
F9-PDOC00397	<b>100.0</b>	40.0	60.0	20.00	40.00
F10-PDOC00443	<b>100.0</b>	85.71	85.71	42.86	71.43
Total	<b>85.11</b>	80.43	67.66	54.47	53.62

Table VI. Accuracy on PROSITE: bold values indicate the best results.

#### 6.4.1 Comparison on Biological Datasets: PROSITE and SCOP.

6.4.1.1 *PROSITE Accuracy Comparison.* Table VI shows the accuracy results on the ten families from PROSITE for VOGUE, HMM, HMMER and  $k^{\text{th}}$ -order HMM. The class specific and total accuracy results are shown, with the best accuracy highlighted in bold. For VOGUE, we set the minimum support at 1, i.e., all patterns that occur even once are used to build the model. The maximum gap is also set to 1 for all classes, except for F5 and F6 which use a gap of 2, and F7 uses a gap of 8. The PROSITE motifs for these classes, especially F7, contain several gaps, necessitating a larger maximum gap value. The number of states for VOGUE is given in Table VII, whereas the number of states in the HMMER model is given in Table VIII.

For the basic HMM we tried  $N = 50, 100, 200$  and found that 100 states worked the best. We then ran the HMM three times, and recorded the best results in the table above. We tried a similar approach to select the number of states for higher order HMMs, namely for 2-HMM (2nd order) and 4-HMM (4th order). We found that for 2-HMM a model with  $N = 5$  states performed the best. It yielded a total accuracy of 54%, whereas increasing the number of states decreased the classification accuracy. For example for  $N = 10$  the accuracy was 53%, for  $N = 20$  it was 51%, for  $N = 50$  it was 49%, and for  $N = 100$  it was only 37%. For 4-HMM, the models with  $N = 5$  and  $N = 10$  gave the same accuracy, with higher number of states yielding worse results; we report results for  $N = 5$ .

We find that in general the higher order HMMs, 2-HMM and 4-HMM, were not able to model the training sequences well. The best accuracy was only 54.47% for 2-HMM and 53.62% for 4-HMM, whereas for the basic 1st order HMM the accuracy was 67.66%. HMMER did fairly well, which is not surprising, since it is specialized to handle protein sequences. It's overall accuracy was 80.43%. VOGUE vastly outperforms the regular HMM. Even more interesting is that VOGUE outperforms HMMER, with VOGUE having an accuracy of 85.11%. This is remarkable when we consider that VOGUE is completely automatic and does not have explicit domain knowledge embedded in the model, except what is recovered from the relationship between the symbols in the patterns via mining.

It is worth noting that for protein sequences, since the alphabet is small (only 20 amino acids), and the sequences are long, all 400 pairs of symbols, i.e., 2-sequences, are frequent, and as such the minimum support does not impact the number of first and second symbol states. Both  $N_1$  and  $N_2$  remain 20 for all the values of minimum support (across all classes); thus we kept the minimum support at 1. That is, all classes have  $N_1 = N_2 = 20$  and the number of gap states is close to  $\approx 400$  ( $G \in [392, 400]$ ), as shown in Table VII. However, the models are not identical, since the frequencies of the various 2-sequences are different, and more importantly, the symbols that appear in the gaps and their frequencies are all different. It is these differences, which are extracted by the VGS method and then modeled by VOGUE, that enable us to discriminate between the classes.

6.4.1.2 *SCOP Accuracy Comparison.* Table IX shows the comparison between VOGUE, HMMER, HMM and 2-HMM, on the 10 test sequences from all the 10 SCOP families. HMM gave the best results for  $N = 20$  states, whereas 2-HMM was run with  $N = 5$  states, since it gave the same results as with  $N = 10, 20$  states. VOGUE was run with a minimum support of 1, but the maximum gap

<b>PROSITE</b>	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
$N_1$	20	20	20	20	20	20	20	20	20	20
$N_2$	20	20	20	20	20	20	20	20	20	20
$G$	394	400	400	400	399	400	400	400	392	400
$N$	435	441	441	441	440	441	441	441	433	441

<b>SCOP</b>	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
$N_1$	20	20	20	20	20	20	20	20	20	20
$N_2$	20	20	20	20	20	20	20	20	20	20
$G$	400	397	398	365	275	390	252	298	380	399
$N$	441	438	439	406	311	431	247	439	421	440

<b>CSLOGS</b>	edu	oth
$N_1$	124	649
$N_2$	201	814
$G$	329	1819
$N$	655	3283

<b>Intrusion</b>	User0	User1	User2	User3	User4	User5	User6	User7	User8
$N_1$	95	103	112	112	183	154	232	176	281
$N_2$	92	100	113	105	191	152	236	168	280
$G$	609	790	847	936	1532	1167	2169	1149	2531
$N$	797	994	1073	1154	1907	1474	2638	1494	3093

<b>Spelling</b>	Incorrect	Correct
$N_1$	105	116
$N_2$	106	124
$G$	785	930
$N$	997	1171

Table VII. VOGUE: Number of States

<b>PROSITE</b>	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
$N$	1049	391	411	897	198	312	88	1157	357	664

<b>SCOP</b>	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
$N$	195	163	142	54	59	83	69	279	177	143

Table VIII. HMMER: Number of States

was set to 30. The number of different types of states in VOGUE are shown in Table VII. The number of states in the HMMER model is given in Table VIII. Unlike the PROSITE dataset, which has a well defined sequence motif per family, the SCOP dataset groups structurally similar proteins together into the families, and there may not be a well defined sequence motif as such. For this reason, we had to increase the max-gap to a larger value to capture longer range dependencies in the model. Note also that due to its structural nature, the SCOP dataset is slightly harder to model via purely sequence based methods like HMMs. The table shows that 2-HMM fared poorly, whereas VOGUE had the the highest classification accuracy of 77.5%. It tied for or had the best class-specific accuracy in 7 out of the 10 classes.

Class	VOGUE	HMMER	HMM	2-HMM
F1-49417	<b>100</b>	25	<b>100</b>	<b>100</b>
F2-46458	<b>100</b>	<b>100</b>	<b>100</b>	75
F3-46626	<b>100</b>	75	75	75
F4-46689	<b>50</b>	<b>50</b>	<b>50</b>	25
F5-46997	25	<b>100</b>	25	0
F6-47095	75	75	<b>100</b>	50
F7-47113	25	<b>100</b>	25	0
F8-48508	<b>100</b>	25	75	0
F9-69118	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>
F10-81296	<b>100</b>	<b>100</b>	75	50
Total	<b>77.5</b>	75.0	72.5	47.5

Table IX. Accuracy on SCOP

Class	VOGUE	HMM
edu	<b>67.39</b>	61.51
oth	84.87	<b>85.04</b>
Total	<b>81.01</b>	79.84

Table X. Accuracy on CSLOGS Dataset

6.4.2 *Comparison on CSLOGS dataset.* On CSLOGS we compare VOGUE with a first order HMM. Since HMMER models only protein sequences, it is not used here. For HMM,  $N = 10$  states worked the best, and the accuracy reported is the best among three runs. For VOGUE, we used a minimum support of 10, and a maximum gap of 1. Table VII shows the number of first and second symbol, gap and total states in the VOGUE model. Table X shows the accuracy results. Overall, VOGUE has a slightly higher accuracy than HMM. However, it is worth noting that VOGUE is significantly better for the minority `edu` class (`edu` comprises 22.1% of the test instances), which is harder to model and classify. HMM has a slight edge on the easier majority `oth` class.

Class	VOGUE	HMM
User0	<b>87.05</b>	71.22
User1	52.89	<b>64.46</b>
User2	<b>77.01</b>	72.19
User3	<b>90.0</b>	<b>90.0</b>
User4	82.30	<b>84.51</b>
User5	<b>73.33</b>	70.37
User6	<b>59.5</b>	55.21
User7	<b>66.97</b>	64.56
User8	88.38	<b>91.41</b>
Total	<b>73.17</b>	71.49

Table XI. Accuracy on Intrusion Dataset

6.4.3 *Comparison on Intrusion Dataset.* Table XI shows the results on the Intrusion dataset. The best baseline HMM model used  $N = 20$  states. VOGUE used minimum support of 2 and maximum gap of 1. The number of different types

of states in VOGUE are shown in Table VII. VOGUE outperformed the baseline HMM method for 6 out of the 9 classes (users), and had a higher overall accuracy too.

Class	VOGUE	HMM
Correct	78.5	<b>78.75</b>
Incorrect	<b>56.27</b>	42.51
Total	<b>68.5</b>	62.45

Table XII. Accuracy on Spelling Dataset

6.4.4 *Comparison on Spelling Dataset.* For HMM, we used  $N = 20$ , and report the best of tree runs. For VOGUE we used minimum support of 10 and minimum gap of 5. Table VII shows the number of different types of states in VOGUE. VOGUE significantly outperforms HMM with an overall accuracy of 68.5%, and moreover, it does much better on the class of more relevance, namely the incorrect spellings, as opposed to the correct spellings.

## 6.5 Timing Comparison

PROSITE	VOGUE	HMMER	HMM	2-HMM	4-HMM
Training	1.21s	38.3s	100.67s	0.76s	3.11s
Testing	2.9s	0.85s	0.51s	0.03s	1.88s

SCOP	VOGUE	HMMER	HMM	2-HMM
Training	0.57s	0.14s	0.58s	0.05s
Testing	3.08s	0.05s	0.03s	0.02s

CSLOGS	VOGUE	HMM
Training	77.56s	1200.01s
Testing	0.03s	0.71s

Intrusion	VOGUE	HMM
Training	14.31s	64.81s
Testing	0.11s	0.12s

Spelling	VOGUE	HMM
Training	44.25s	25.76s
Testing	0.38s	0.69s

Table XIII. Run Times: PROSITE, SCOP, CSLOGS, Intrusion and Spelling

In Table XIII, we show the execution time for building and testing the different HMM models on the various datasets. The training times are the average across all classes, and the testing times give the average time per test sequence for the viterbi algorithm. For example, for PROSITE the time reported for training is the average time per family taken over all the ten families. The time for testing is the average time per test sequence taken over all the 235 test sequences run against the ten families (i.e., average over 2350 runs). The time for VOGUE includes the

mining by VGS, and for HMMER, the alignment by CLUSTAL-W. VOGUE builds the models much faster than HMMER (over 30 times), since the time taken by CLUSTAL-W for multiple sequence alignment dominates the overall training time for HMMER. On the other hand, HMMER's testing time is more than twice as fast as VOGUE; this is because HMMER is essentially a left-to-right model, whereas VOGUE also models durations. The training time for the basic HMM is high since the best model had 100 states and the Baum-Welch (or expectation-maximization) training over a large number of states is slow. Once the model is built, the viterbi algorithm runs very fast. Finally, the training times for 2-HMM and 4-HMM are very low, since the model only has 5 states.

The training and testing times on the SCOP dataset are very small, since there are only 6 sequences to train over, and 4 sequences to test over, for each of the models. The regular HMM used only 20 states and 2-HMM only 5 states, thus they train very quickly. On the web usage logs, even though there were only 10 states in the HMM model, the Baum-Welch training took a very long time, due to the large number (16175; see Table V) of training sequences. VOGUE was over 15 times faster during training, and was also faster for testing. For the Intrusion dataset the best HMM model had 20 states, but since there were large number of training sequences (between 364 and 1280) per class, the Baum-Welch algorithm took longer time to converge. VOGUE is over 4 times faster during model building, since mining the 2-sequences in VGS is fairly fast. On Spelling, the HMM model with 20 states is about 1.7 times faster than VOGUE.

Overall we conclude that as expected the training times depend on the number of training sequences and states in the model. For HMMER, the time is dominated by the multiple sequence alignment step. For HMM and high-order HMMs, the time is dominated by the Baum-Welch training method. For VOGUE the main time is spent in the VGS method. In general the VGS mining is much faster than the alignment or Baum-Welch steps, especially when there are many training sequences. For testing, all methods are generally very efficient.

## 7. CONCLUSIONS AND FUTURE WORK

VOGUE combines two separate but complementary techniques for modeling and interpreting long range dependencies in sequential data: pattern 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 automatically model multiple higher order HMMs via a single variable-order model that executes faster and yields much greater accuracy than the state-of-the-art techniques. For data decoding and interpretation, we designed an optimized Viterbi algorithm that exploits the fact that the transition matrix between the states of our model is sparse, so there is no need to model the transitions between all the states.

We applied VOGUE to finding homologous proteins in the given database of  
ACM Journal Name, Vol. X, No. X, XX 2009.

protein sequences. The goal is to build a statistical model to determine whether a query protein belongs to a given family (class) or not. We show experimentally that on this dataset VOGUE outperforms HMMER [Eddy 1998], a HMM model especially designed for protein sequences. Likewise VOGUE outperforms higher-order HMMs, and the regular HMM model on other real sequence data taken from web usage logs at the CS department at RPI, user command sequences for masquerade and intrusion detection, and text data for spelling correction.

In the future, we plan to apply VOGUE to other interesting and challenging real world problems. We also would like to implement and test the generalized VOGUE model for sequences of longer length. Finally, we would like to develop alternate variable length and variable duration HMM models, with the aim to further improve the modeling capabilities of VOGUE.

#### ACKNOWLEDGMENTS

We would like to thank Bouchra Bouqata for her work on the preliminary version of this paper [Bouqata et al. 2006]. This work was supported in part by NSF Grants EMT-0829835, and CNS-0103708, and NIH Grant 1R01EB0080161-01A1.

#### REFERENCES

- AGRAWAL, R. AND SRIKANT, R. 1995. Mining sequential patterns. In *Intl. Conf. on Data Engg.*
- ANTUNES, C. AND OLIVEIRA, A. L. 2003. Generalization of pattern-growth methods for sequential pattern mining with gap constraints. *Machine Learning and Data Mining in Pattern Recognition, Springer LNCS Vol. 2734*, 239–251.
- BOUQATA, B., CAROTHERS, C. D., SZYMANSKI, B. K., AND ZAKI, M. J. 2006. VOGUE: A novel variable order-gap state machine for modeling sequences. In *10th European Conference on Principles and Practice of Knowledge Discovery*.
- BUHLMANN, P. AND WYNER, A. 1999. Variable length markov chains. *Annals of Statistics* 27, 480–513.
- CHAOJI, V., HOONLOR, A., AND SZYMANSKI, B. 2008. Recursive data mining for role identification. In *5th International Conference on Soft Computing as Transdisciplinary Science and Technology*, 218–225.
- DESHPANDE, M. AND KARYPIS, G. 2001. Selective markov models for predicting web-page accesses. In *SIAM International Conference on Data Mining*.
- DONG, G. AND PEI, J. 2007. *Sequence Data Mining*. Springer.
- DU PREEZ, J. 1998. Efficient training of high-order hidden markov models using first-order representation. *Computer Speech and Language* 12(1), 23–39.
- DURBIN, R., EDDY, S., KROGH, A., AND MITCHISON, G. 1998. *Biological Sequence Analysis*. Cambridge University Press.
- EDDY, S. 1998. Profile hidden markov models. *Bioinformatics* 14, 9, 755–63.
- FALQUET, L., PAGNI, M., BUCHER, P., HULO, N., SIGRIST, C. J. A., HOFMANN, K., AND BAIROCH, A. 2002. The PROSITE database, its status in 2002. *Nucleic Acids Research* 30, 1, 235–238.
- FELZENSZWALB, P., HUTTENLOCHER, D., AND KLEINBERG, J. 2003. Fast algorithms for large-state-space HMMs with applications to web usage analysis. In *Advances in Neural Information Processing Systems*.
- FINE, S., SINGER, Y., AND TISHBY, N. 1998. The hierarchical hidden markov model: Analysis and applications. *Machine Learning* 32, 41–62.
- GALASSI, U., BOTTA, M., AND GIORDANA, A. 2007. Hierarchical hidden markov models for user/process profile learning. *Fundamenta Informaticae* 78, 4, 487–505.
- GAROFALAKIS, M., RASTOGI, AND SHIM, K. 2002. Mining sequential patterns with regular expression constraints. *IEEE Transactions on Knowledge and Data Engineering* 14, 530–552.

- GOLDING, A. AND ROTH, D. 1996. Applying winnow to context-sensitive spelling correction. In *Proc. 13th Int. Conf. Machine Learning*. 180–190.
- GUSFIELD, D. 1997. *Algorithms on strings, trees, and sequences: Computer science and computational biology*. Cambridge University Press.
- JENSEN, K., STYCZYNSKI, M., RIGOUTSOS, I., AND STEPHANOPOULOS, G. 2006. A generic motif discovery algorithm for sequential data. *Bioinformatics* 22, 21–28.
- KRIOUILE, A., MARI, J.-F., AND HAON, J.-P. 1990. Some improvements in speech recognition algorithms based on HMM. In *International Conference on Acoustics, Speech, and Signal Processing*.
- KUCERA, H. AND FRANCIS, W. 1967. *Computational Analysis of Present-Day American English*. Brown University Press, Providence, RI.
- LANE, T. AND BRODLEY, C. 1999. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security* 2, 295–331.
- LAW, H. H.-C. AND C.CHAN. 1996. N-th order ergodic multigram hmm for modeling of languages without marked word boundaries. In *International Conference On Computational Linguistics*.
- LAXMAN, S., SASTRY, P., AND UNNIKRISHNAN, K. P. 2005. Discovering frequent episodes and learning hidden markov models: A formal connection. *IEEE Transactions on Knowledge and Data Engineering* 17, 11 (Nov), 1505–1517.
- LI, C. AND WANG, J. 2008. Efficiently mining closed subsequences with gap constraints. In *SIAM International Conference on Data Mining*.
- MANNILA, H., TOIVONEN, H., AND VERKAMO, I. 1995. Discovering frequent episodes in sequences. In *1st Intl. Conf. Knowledge Discovery and Data Mining*.
- MURZIN, A., BRENNER, S., HUBBARD, T., AND CHOTHIA, C. 1995. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.* 247, 4 (apr), 536–540.
- NANOPOULOS, A., KATSAROS, D., AND MANOLOPOULOS, Y. 2003. A data mining algorithm for generalized web prefetching. *IEEE Transactions on Knowledge and Data Engineering* 15(5), 1155–1169.
- PEI, J., HAN, J., MORTAZAVI-ASL, B., PINTO, H., DAYAL, Q. C. U., AND HSU, M.-C. 2001. PrefixSpan: Mining sequential patterns efficiently by prefix projected pattern growth. In *International Conference on Data Engineering*.
- PITKOW, J. AND PIROLI, P. 1999. Mining longest repeating subsequence to predict WWW surfing. In *2nd USENIX Symp. on Internet Technologies and Systems*.
- RABINER, L. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE* 2, 77, 257–286.
- RON, D., SINGER, Y., AND TISHBY, N. 1996. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning* 25, 2-3, 117–149.
- SAUL, L. AND JORDAN, M. 1999. Mixed memory markov models: Decomposing complex stochastic processes as mix of simpler ones. *Machine Learning* 37, 1, 75–87.
- SCHWARDT, L. C. AND DU PREEZ, J. A. 2000. Efficient mixed-order hidden markov model inference. In *International Conference on Spoken Language Processing*.
- SRIKANT, R. AND AGRAWAL, R. 1996. Mining sequential patterns: Generalizations and performance improvements. In *5th Intl. Conf. Extending Database Technology*.
- SZYMANSKI, B. AND ZHANG, Y. 2004. Recursive data mining for masquerade detection and author identification. In *5th IEEE System, Man and Cybernetics Information Assurance Workshop*. 424–431.
- WANG, Y., ZHOU, L., FENG, J., WANG, J., AND LIU, Z.-Q. 2006. Mining complex time-series data by learning markovian models. In *6th IEEE International Conference on Data Mining*.
- ZAKI, M. J. 2000. Sequences mining in categorical domains: Incorporating constraints. In *9th Intl. Conf. on Information and Knowledge Management*.
- ZAKI, M. J. 2001. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal* 42, 1/2 (Jan/Feb), 31–60.
- ACM Journal Name, Vol. X, No. X, XX 2009.

- ZHANG, M., KAO, B., CHEUNG, D. W.-L., AND YIP, K. 2005. Mining periodic patterns with gap requirement from sequences. In *ACM SIGMOD Intl. Conf. Management of Data*.
- ZHU, X. AND WU, X. 2007. Mining complex patterns across sequences with gap requirements. In *International Joint Conference on AI*.

Received Month Year; revised Month Year; accepted Month Year