CLUSEQ: Efficient and Effective Sequence Clustering

Jiong Yang
UIUC
jioyang@cs.uiuc.edu

Wei Wang
UNC Chapel Hill
weiwang@cs.unc.edu

Abstract

Analyzing sequence data has become increasingly important recently in the area of biological sequences, text documents, web access logs, etc. In this paper, we investigate the problem of clustering sequences based on their structural features. As a widely recognized technique, clustering has proven to be very useful in detecting unknown object categories and revealing hidden correlations among objects. One difficulty that prevents clustering from being performed extensively on sequence data (in categorical domain) is the lack of an effective yet efficient similarity measure. Therefore, we propose a novel model (CLUSEQ) for sequence cluster by exploring significant statistical properties possessed by the sequences. The conditional probability distribution (CPD) of the next symbol given a preceding segment is derived and used to characterize sequence behavior and to support the similarity measure. A variation of the suffix tree, namely probabilistic suffix tree, is employed to organize (the significant portion of) the CPD in a concise way. A novel algorithm is devised to efficiently discover clusters with high quality and is able to automatically adjust the number of clusters to its optimal range via a unique combination of successive new cluster generation and cluster consolidation. The performance of CLUSEQ has been demonstrated via extensive experiments on several real and synthetic sequence databases.

1 Introduction

During recent years, analyzing sequence data (particularly in categorical domains) has become increasingly important, partially due to the significant advances in the biological study as well as many other fields. Examples of sequence data include genomic DNA sequences, unfolded protein sequences, text documents, web usage data, system traces, etc.. Previous work on mining sequence data has mainly focused on the frequent pattern discovery. In this paper, we focus on the problem of clustering sequence data.

Clustering has been widely recognized as a powerful data mining technique and has been studied extensively during recent years. The major goal of clustering is to create a partition of objects such that objects in each group have similar features. The result can potentially reveal unknown object groups/categories that may lead to a better understanding of the nature. In the context we address, each object is a symbol sequence and all potential features of a sequence are encoded (implicitly) in the specific symbol layout in the sequence. It is interesting to notice that these structural characteristics sometimes can uniquely determine the functional properties of the sequence and often plays a decisive role in meaningful clustering of the sequence.

- A protein sequence is an ordered list of amino acids. The functionality of the protein sequence is solely encoded in the specific ordering of the amino acids. Protein sequences with similar biological functions would share some common signature (e.g., conserved protein regions).
- A text document is essentially a (long) list of alphabet in a certain order and its semantic can be inferred by the spelling of each word and the order of words.

Both of the above examples suggest that clustering based on sequential structural characteristics can serve as a powerful tool to discriminate sequences belonging to different functional categories. Significant challenges exist on how to abstract and represent distinctive structural characteristics (of sequences of vastly different lengths) and to design an effective yet efficient similarity measure based on them. One possible approach to tackle this difficulty is to use the *edit distance* [11] to measure the distance between each pair of sequences. This, however, is not an ideal solution because, in addition to its inefficiency in calculation, the edit distance only captures the optimal global alignment between a pair of sequences, but ignores many other local alignments that often represent important features shared by the pair of sequences[1]. These overlooked features may be very crucial to produce meaningful clusters. Even though allowing *block operations*[2] [19, 21] may alleviate this weakness to a certain degree, the

[1] Consider three sequences $aaaabbb$, $bbbaaaa$, and $abcdefg$. The edit distance between $aaaabbb$ and $bbbaaaa$ is 6 whereas the edit distance between $aaaabbb$ and $abcdefg$ is also 6. This, to a certain extent, contradicts the intuition that $aaaabbb$ is more similar to $bbbaaaa$ than $abcdefg$.

[2] A consecutive block can be inserted/deleted/shifted/reversed in a sequence with a constant cost with regard to the edit distance.

computation of edit distance with block operations is N-P hard [21]. This limitation of edit distance, in part, has motivated researchers to explore alternative solutions.

Another approach that has been widely used in document clustering is the keyword-based method. Instead of being treated as a sequence, each text document is regarded as a set of keywords or phrases and is usually represented by a weighted word vector. The similarity between two documents is measured based on keywords and phrases shared by them and is often defined in some form of normalized dot-product [12, 17, 24, 25, 29]. A direct extension of this method to generic symbol sequences is to use short segments of fixed length q (generated using a sliding window through each sequence) as the set of "words" in the similarity measure. This method is also referred to as the q-gram based method in the literature[8, 22, 26, 27]. While the q-gram based approach enables significant segments (i.e., keywords/phrases/q-grams) to be identified and used to measure the similarity between sequences regardless of their relative positions in different sequences, valuable information may be lost as a result of ignoring sequential relationship (e.g., ordering, correlation, dependency, etc.) among these segments, which impacts the quality of clustering.

In this paper, we propose a novel model for sequence cluster by exploring significant patterns of sequence formation. Statistics properties of sequence construction are used to assess the similarity. Sequences belonging to one cluster may subsume to the same probability distribution of symbols (conditioning on the preceding segment of a certain length), while different clusters may follow different underlying probability distributions. This feature, typically referred to as *short memory*, which is common to many applications, indicates that, for a certain sequence, the empirical probability distribution of the next symbol given the preceding segment can be accurately approximated by observing no more than the last L symbols in that segment. Significant features of such probability distribution can be very powerful in distinguishing different clusters. By extracting and maintaining significant patterns characterizing (potential) sequence clusters, one can easily determine whether a sequence should belong to a cluster by calculating the likelihood of (re)producing the sequence under the probability distribution that characterizes the given cluster.

To support efficient maintenance and retrieval of the probability entries[3], a novel variation of the suffix tree [9, 11, 23, 28], namely the *probabilistic suffix tree* (PST), is employed to serve as a compact representation to organize the derived (conditional) probability distribution for a cluster of sequences. A probability vector is associated with each node to store the probability distribution of the next symbol given the label of the node as the preceding segment. These innovations enable the similarity estimation to be performed very fast, which offers many advantages over alternative methods and plays a dominant role in the overall performance of our clustering algorithm. A performance comparison with existing methods is furnished in Section 6.

In many applications, neither the number of clusters in the data set nor the percentage of outliers is known in advance. To solve this problem, we devise a novel clustering algorithm (CLUSEQ) that produces a set of possibly overlapped clusters and is able to automatically adjust the number of clusters and the boundary to separate clustered sequences from outliers, benefiting from a unique combination of successive new cluster generation and cluster consolidation.

In summary, we claim the following contributions in this paper.

- A novel model of sequence cluster is introduced, which can handle sequences with various lengths seamlessly. The conditional probability distribution of the next symbol given a preceding segment is used as a means to characterize sequence behavior and as a foundation for the similarity measurement between sequences.

- A novel variation of the suffix tree family, namely *probabilistic suffix tree*, is employed to efficiently organize statistical properties of a sequence cluster.

- A novel algorithm CLUSEQ is designed to cluster sequences into a set of possibly overlapped clusters, where the number of clusters and the amount of outliers can be automatically adjusted during the clustering process.

The remainder of this paper is organized as follows. Section 2 gives the formal definition of the problem we try to solve. The probabilistic suffix tree structure is provided in Section 3. Section 4 presents the CLUSEQ algorithm while Section 5 discusses some implementation issues. The experimental results are shown in Section 6. Some related work is surveyed in Section 7 and the conclusion is drawn in Section 8.

2 Sequence Cluster

We now formalize the problem that we try to solve in this paper. Let $\Im = \{s_1, s_2, \ldots, s_n\}$ be the set of all possible symbols. A **sequence** is an ordered list of symbols in $\Im$. The number of symbols in a sequence is referred to as the **length** of sequence. Given a sequence, a **segment** is defined as a *consecutive* portion of the sequence. For example, "bcd" is a segment of "abcdef" while "abd" is not. Conventionally, we use the term "sequence" to refer to a whole symbol sequence in the database while the term "segment" to denote a portion of some sequence. A **sequence database** is a set of sequences. Given a sequence database, our objective is to categorize these sequences into clusters according to their structural similarities.

As we explained previously, both the edit distance and the q-gram based method are not desirable choices to measure the similarity. In this paper, we build the similarity measure upon statistical properties of the sequences, which overcomes

[3]Even though the hidden Markov model can be used for this purpose, its computational inefficiency prevents it from being applied to a large dataset.

drawbacks of previous methodologies. More precisely, the conditional probability distribution (CPD) of the next symbol right after some preceding segment is employed to represent the structural properties of a single sequence or a set of sequences. One way to evaluate the similarity between two sequences or among a set of sequences is to compute the difference between the corresponding conditional probability distributions. There have been many methods to assess the difference between two probability distributions, among which the *variational distance* and the *Kullback-Leibler divergence* are the most popular ones [18]. Given two probability distributions (P_1 and P_2) of the variable σ, the variational distance is defined as $V(P_1, P_2) = \sum_{\sigma \in \Omega} |P_1(\sigma) - P_2(\sigma)|$ while the Kullback-Leibler divergence is defined as $J(P_1, P_2) = I(P_1, P_2) + I(P_2, P_1) = \sum_{\sigma \in \Omega} (P_1(\sigma) - P_2(\sigma)) \log \frac{P_1(\sigma)}{P_2(\sigma)}$, where Ω is the domain of σ. In our case, σ represents a segment and Ω is the set of all possible segments (up to a certain length). A common theme shared by these two methods is that the difference between two probability distributions is measured as an aggregation of difference defined on each possible segment (up to a certain length). The longer the segment considered in the computation, the more accurate the difference measure. If we consider segments up to length L in above formulas, then there are $O(|\Im|^L)$ distinct segments (i.e., $|\Omega| = O(|\Im|^L)$). The computational complexity of calculating the difference between two probability distributions is exponential with respect to the length of the segment σ. This is very time consuming when the segment length is reasonably long, which is typically the case in the problems in which we are interested. To avoid the expensive distance computation, we employ an alternative method. The key idea is that, given a sequence cluster S and the conditional probability distribution P modeling it, a sequence should subsume to a similar conditional probability distribution if the sequence can be *predicted* under P with relatively high probability. The probability to predict a sequence $\sigma = s_1 s_2 \ldots s_l$ is

$$P_S(\sigma) = P_S(s_1) \times P_S(s_2|s_1) \times \ldots \times P_S(s_l|s_1 \ldots s_{l-1})$$

where $P_S(s_i|s_1 \ldots s_{i-1})$ is the conditional probability that the symbol s_i is the next symbol right after the segment $s_1 s_2 \ldots s_{i-1}$ in the sequence cluster S. To reduce the potential influence of noise to the probability estimation, the above estimation is used only when the count of $s_1 \ldots s_{i-1}$ exceeds some threshold c (to reach certain statistical significance). c is referred to as the **significance threshold**. Given a set of sequences, a segment is called a **significant segment** if it appears at least c times, and is called an **insignificant segment** otherwise. (As a rule of thumb, c is usually set to a value greater or equal to 30.) If $s_1 \ldots s_{i-1}$ is insignificant, the *longest significant suffix*[4] $s_j \ldots s_{i-1}$ will be used in the estimation. The value of $P_S(s_i|s_j \ldots s_{i-1})$ is taken to approximate the value of $P_S(s_i|s_1 \ldots s_{i-1})$.

The predict probability $P_S(\sigma)$ can serve as an indicator of the similarity between the sequence σ and the cluster S. If the value of $P_S(\sigma)$ is significantly higher than the probability of predicting/generating σ by a memoryless random process, then we may conclude that the sequence σ subsumes a similar CPD to that of S and may be considered a member of S. The probability $P^r(\sigma)$ that σ is generated by a memoryless random generator is $P^r(\sigma) = \Pi_{i=1}^{l} p(s_i)$ where $p(s_i)$ is the probability of observing the symbol s_i at any given position of any sequence in the database. The **similarity** between the sequence σ and the cluster S is then defined as $sim_S(\sigma) = \frac{P_S(\sigma)}{P^r(\sigma)} = \frac{\Pi_{i=1}^{l} P_S(s_i|s_1 \ldots s_{i-1})}{\Pi_{i=1}^{l} p(s_i)} = \Pi_{i=1}^{l} \left(\frac{P_S(s_i|s_1 \ldots s_{i-1})}{p(s_i)} \right)$. This similarity measure requires that the entire sequence σ follows a single CPD, which may not always be true. Sometimes, different portions of a sequence may subsume to different CPDs, especially when the sequence is long. (For example, a protein may belong to multiple domains.) To accommodate this scenario, the above similarity measure is modified to capture the *maximum* similarity between any continuous segment of σ and S.

$$SIM_S(\sigma) = \max_{1 \le j \le i \le l} sim_S(s_j \ldots s_i). \tag{1}$$

Note that this similarity measure is not a metric because a sequence may have complex features and may be similar to multiple clusters (or sequences) that could be dissimilar from each other. $SIM_S(\sigma) > 1$ indicates that there is some evidence to show the sequence σ subsumes the CPD of S; and the higher the value of $SIM_S(\sigma)$, the stronger the evidence. If a sequence σ produces a small $SIM(\sigma)$ (e.g., less than 1) for every cluster, then σ is deemed to be an outlier. A threshold t ($t \ge 1$) is employed to separate clustered sequences from the rest. t is referred to as the **similarity threshold** in the remainder of this paper. If the value of $SIM_S(\sigma)$ exceeds t, we may think that σ has sufficiently high similarity to the cluster S.

This leads to our definition of sequence cluster, which requires each sequence in a cluster to be predicted with high probability from the CPD of the cluster.

Definition 2.1 *A set of sequences S is a* **sequence cluster** *if, for each sequence σ in S, the similarity $SIM_S(\sigma)$ between σ and S is greater than or equal to some threshold t.*

Intuitively, t should be set to a value greater than 1 in order to provide a meaningful separation between clustered sequences and outliers. In practice, the proper value of t can be either specified by the user or adjusted by the algorithm during the mining process[5]. Given a sequence database, our objective is to group these sequences into a set of possibly overlapping clusters. Before we formally present the clustering algorithm, we first give some terminology used in the fol-

[4]Given a segment $s_1 \ldots s_{i-1}$, a suffix $s_j \ldots s_{i-1}$ is called the **longest significant suffix** if $s_j \ldots s_{i-1}$ is significant and any longer suffix $s_{j'} \ldots s_{i-1}$ is insignificant where $j' < j$. By definition, the longest significant suffix of a significant segment is always the segment itself.

[5]We'll discuss how to adjust t later in this paper.

lowing discussion and describe the data structure employed during the clustering process.

3 Probabilistic Suffix Tree

Our similarity measure utilizes the conditional probability distribution derived from sequences. The efficiency of the computation relies on the efficient retrieval of conditional probability entries during the sequence prediction. The suffix tree (or its variation) has been proved to be a very successful model to capture and organize significant patterns in symbol sequences. With regard to the problem addressed in this paper, the suffix tree structure can be utilized to serve as the compact indexing structure to organize significant segments and associated conditional probability entries of each cluster.

The model of suffix tree [11, 28] was originally proposed to index segments. A segment σ' of length l' is called a **suffix** of a sequence σ of length l ($l' \leq l$) if $\sigma'[i] = \sigma[i + l - l']$ for $i = 1, 2, \ldots, l'$. σ' is also called a **proper suffix** of σ if $l' < l$. Similarly, a segment σ' of length l' is called a **prefix** of a sequence σ of length l ($l' \leq l$) if $\sigma'[i] = \sigma[i]$ for $i = 1, 2, \ldots, l'$. Again, σ' is also called a **proper prefix** of σ if $l' < l$. For example, a, ab, aba, and $abab$ are prefixes of $abab$, whereas $abab$, bab, ab, and b are suffixes of $abab$. In the remainder of this paper, we sometime omit the word "proper" if no ambiguity incurs.

A traditional **suffix tree** for a set of sequences (over alphabet $\Im$) is a rooted directed tree where each leaf represents a distinct suffix of some sequence in the set. Each edge is labeled with a (short) segment appearing in the sequence set. A key feature of the suffix tree is that, for each node, the concatenation of the edge-labels on the path between the root and the node exactly spells out a distinct segment appearing in the sequence set. This segment is often referred to as the **label** of the node. This feature makes the suffix tree a desirable carrier to organize CPD entries for each sequence cluster. By definition, the estimation of the similarity between a sequence and a cluster consists of a series of retrieval of CPD entries, which often involves the step of locating the longest significant suffix (of some given segment)[6]. To facilitate the process of locating the longest significant suffix and retrieving its associated CPD entries, we build the suffix tree on the *reversed* sequences (instead of the original sequences). Given a sequence $\sigma = s_1 s_2 \ldots s_l$, the reversed sequence is $reversed(\sigma) = s_l \ldots s_2 s_1$ obtained by concatenating symbols in σ in reverse order. An example is shown in Figure 1. Each node is labeled with a distinct segment that can be generated by concatenating edge-labels on the path *from the node to the root*.

A count C is associated with each node to record the number of occurrences of its label in the sequence cluster. (The count associated with the root records the overall size of the sequence cluster, which is equal to the sum of the lengths of every sequence in the cluster). A node is called a **significant node** if its count has a value greater than or equal to c, and is called an **insignificant node** otherwise. It is obvious that the label of a significant node is always a significant segment and the label of an insignificant node is always an insignificant segment. The number inside each node in Figure 1 is the count of the node. The dash line separates the set of significant nodes from the rest if $c = 65$. A probability distribution vector $P(s_i|\sigma')$ (over the alphabet $\Im = \{s_1, s_2, \ldots, s_n\}$) is also associated with each node to record the conditional probability distribution of the next symbol, say s_i, given the label σ' of the node as the preceding segment. The tuple $(0.406, 0.594)$ at node z in Figure 1 represents that the conditional probabilities $P(a|ba)$ and $P(b|ba)$ are 0.406 and 0.594, respectively. This tree is called the **probabilistic suffix tree** to distinguish it from the ordinary suffix tree built on the original sequences without the associated probability vectors.

With the probabilistic suffix tree on hand, the estimation of the conditional probability $P(s_i|s_1 \ldots s_{i-1})$ can be performed very efficiently through a unified procedure, no matter whether the segment $s_1 \ldots s_{i-1}$ is significant or not, which includes two steps. (1) Locate the node whose label $s_j \ldots s_{i-1}$ is the longest significant suffix of $s_1 \ldots s_{i-1}$. This node is referred to as the **prediction node** of the segment $s_1 \ldots s_{i-1}$ and can be located by traversing from the root along the path $\rightarrow s_{i-1} \rightarrow \ldots \rightarrow s_2 \rightarrow s_1$ until we reach either the node labeled with $s_1 s_2 \ldots s_{i-1}$[7] or a significant node where any further advance (along the path) would reach an insignificant node[8]. (2) Retrieve the entry corresponding to the symbol s_i in the probability vector stored at the prediction node. The value of this entry is used as the estimated value of $P(s_i|s_1 \ldots s_{i-1})$. For example, The value of $P(a|bba)$ can be estimated as follows. Starting at the root of the tree in Figure 1, we traverse along the path $\rightarrow a \rightarrow b \rightarrow b$ and stop at node z since any further advance would reach an insignificant node. Node z is the prediction node of the segment bba. (The longest significant suffix of bba is ba.) The probability entry stored at node z is used to approximate the value of $P(a|bba)$. That is, $P(b|bba) \approx P(b|ba) = 0.594$. The computational complexity of estimating $P(s_i|s_1 \ldots s_{i-1})$ is $O(i)$.

4 The CLUSEQ Algorithm

Our proposed clustering algorithm, CLUSEQ, uses a probabilistic suffix tree to store significant features (i.e., CPD) of each sequence cluster. The flowchart of CLUSEQ is presented in Figure 2. The CLUSEQ algorithm takes a sequence database Σ together with three parameters k, c, t as the input and produces a set of clusters. At the beginning, all sequences in the database are unclustered. The general idea of CLUSEQ is that, starting from the set of unclustered

[6]This is because, when the preceding segment is long, it is very likely that this segment is insignificant, which requires the CPD on the longest significant suffix to be used as an approximation of the original CPD.

[7]This happens when $s_1 s_2 \ldots s_{i-1}$ is a significant segment.

[8]This is the scenario if $s_1 s_2 \ldots s_{i-1}$ is an insignificant segment.

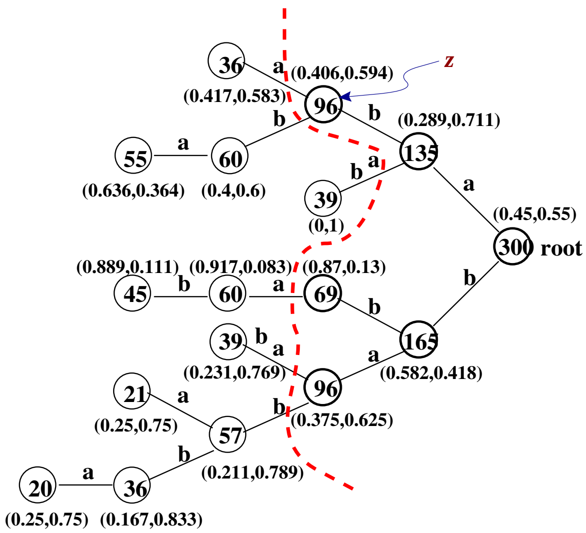

Figure 1. A Probabilistic Suffix Tree

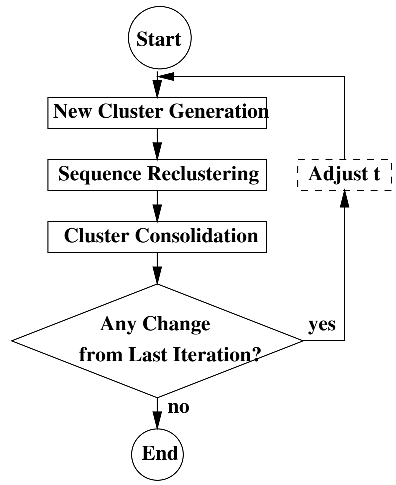

Figure 2. The Flowchart of the CLUSEQ Algorithm

sequences, an iterative process is employed to continuously improve the quality of the clustering until no further improvement can be made. During each iteration, a set of new clusters are generated from the set of unclustered sequences to augment the current set of clusters, which is followed by a sequential examination of every sequence to evaluate its similarity to each cluster and to update its cluster membership. At the end of each iteration, a consolidation procedure is invoked to merge heavily overlapped clusters. An optional step can also be taken before the start of the next iteration to adjust the similarity threshold t (if necessary). The entire process terminates when the clustering produced by the current iteration remains the same as that of the previous iteration. (That is, the number of clusters is the same and no sequence changes membership.) We now discuss each step in detail in the following subsections.

4.1 New Cluster Generation

A unique feature of the CLUSEQ algorithm is that new clusters are initiated and added to the current collection of clusters at the beginning of every iteration. The purpose of the successive generation of new clusters from unclustered sequences is to adapt the number of clusters to its appropriate range. This step, coupled with the cluster consolidation, enables the CLUSEQ algorithm to adapt to the appropriate number of clusters during the course of clustering, regardless of the number of initial clusters used.

At the first iteration, all sequences are unclustered and k new clusters are selected to serve as the initial clusters. (The default value of k is 1.) During each subsequent iteration, the number of new clusters generated is $k' \times f$ where k' and f ($0 \le f \le 1$) are the number of existing clusters at the end of the previous iteration and a growth factor, respectively. The growth factor controls the pace of the increment of the number of clusters and is defined as $f = \frac{\max\{k'_n - k'_c, 0\}}{k'_n}$ where k'_n and k'_c are the number of new clusters generated at the beginning of the previous iteration and the number of clusters eliminated as a result of cluster consolidation at the end of the previous iteration, respectively. The rationale is that, if the current number of clusters is far below the actual number of clusters, the consolidation step is unlikely to reduce the number of clusters on a large scale and hence the value of f is close or equal to 1, which allows the number of clusters to increase at an exponential pace. When the current number of clusters approaches to or even exceeds the actual number of clusters, it is very likely that a large number of clusters are removed during the consolidation step, which keeps the generation of new clusters at a much less ambitious pace.

Each new cluster at its initial stage contains only one sequence and is represented by the probabilistic suffix tree constructed from the sequence. To generate k_n new clusters, a set of k_n unclustered sequences need to be chosen as the seeds. Ideally, each new cluster should have as little similarity to existing clusters and other new clusters as possible. A straightforward way to achieve this goal is to compute pairwise similarity between every pair of unclustered sequences and between every unclustered sequence and every existing cluster, which is very inefficient if there is a large number of unclustered sequence. (Note that all sequences are unclustered at the beginning of the first iteration.) To expedite the process, we employ a sampling technique and restrict the scope of seed selection to the set of sample sequences. At the beginning, a set of m unclustered sequences $S_1, S_2, \ldots, S_m$ are selected randomly where $m \geq k_n$. For each sample sequence S_i, a probabilistic suffix tree PST_i is constructed. From these m probabilistic suffix trees, k_n trees will be chosen as the seeds. The following heuristics can be used to choose the optimal seeds. Let T be the set of existing clusters. A greedy algorithm is carried out, which consists of k_n steps. At each step, each (remaining) sample sequence is examined to calculate the highest similarity to any cluster in T, and among them, the sequence with the least similarity to all clusters in T is selected and put in T. At the end of this procedure, k_n new clusters have been added to T.

In this paper, we set $m = 5k_n$ unless otherwise specified. With $5k_n$ samples, it can deliver a satisfactory result without a prolonged process. The experimental results in Section 6 confirm with our conjecture. The computational complexity of the new cluster generation process is $O(m \times (m + k') \times l^2) = O(k_n \times (k_n + k') \times l^2)$ where k_n, k', and l are the number of new clusters, the number of existing clusters, and the average sequence length, respectively.

4.2 Sequence Reclustering

In this step, each sequence is examined against every cluster to identify the similar one(s). A sequence σ will join a cluster if its similarity is above the threshold t. For each cluster that σ joins, the segment that produces the maximum similarity score will be used to update the probabilistic suffix tree of that cluster. If a sequence has low similarity to every cluster, it would remain unclustered.

4.3 Similarity Estimation

The similarity estimation is very crucial to both the accuracy and efficiency of the CLUSEQ algorithm. Given a probabilistic suffix tree PST that models the cluster S, the similarity of a sequence $\sigma = s_1 s_2 \ldots s_l$ to S is defined by Equation 1. A dynamic programming method can be used to calculate $SIM_S(\sigma)$ via a single scan of σ. Let

$$X_i = \frac{P_S(s_i | s_1 \ldots s_{i-1})}{p(s_i)},$$

$$Y_i = \max_{1 \le j \le i} sim_S(s_j \ldots s_i),$$

$$Z_i = \max_{1 \le i1 \le i2 \le i} sim_S(s_{i1} \ldots s_{i2}).$$

Intuitively, X_i, Y_i, and Z_i can be viewed as the similarity contributed by the symbol on the ith position of the sequence, the maximum similarity possessed by any segment ending at the ith position, and the maximum similarity possessed by any segment ending prior to or on the ith position, respectively. Then, $SIM_S(\sigma) = Z_l$, which can be obtained by

$$Y_i = \max\{Y_{i-1} \times X_i, X_i\},$$

$$Z_i = \max\{Z_{i-1}, Y_i\},$$

and $Y_1 = Z_1 = X_1$. The value of $P_S(s_i | s_1 s_2 \ldots s_{i-1})$ can be retrieved from the prediction node of $s_1 s_2 \ldots s_{i-1}$ in PST. Consider the sequence *bbaa* and the probabilistic suffix tree in Figure 1. Table 1 shows the procedure of calculating the similarity between them, assuming that the probabilities of observing a and b in the entire sequence database are

Table 1. Similarity Estimation of $bbaa$

Sequence	***b***	***b***	***a***	a
$P_S(s_i \vert s_1 \ldots s_{i-1})$	**0.55**	**0.418**	**0.87**	0.406
X_i	**1.38**	**1.05**	**1.45**	0.677
Y_i	**1.38**	**1.45**	**2.10**	1.42
Z_i	1.38	1.45	**2.10**	**2.10**

0.6 and 0.4, respectively. (That is, $p(a) = 0.6, p(b) = 0.4$.) In this example, the segment *bba* produces the maximum similarity, which is 2.10. The computational complexity of estimating the similarity is $O(l^2)$ where l is the length of the sequence. Nevertheless, the actual computation time is significantly below this theoretical bound. With the help of some additional structure (e.g., auxiliary links), the computational complexity could be reduced to $O(l)$. Due to the space limitations, we will not discuss it in detail.

4.4 Maintenance of the Probabilistic Suffix Tree

When a new cluster is initialized, it only contains one sequence that is used to build the initial probabilistic suffix tree for this cluster. Given a sequence $\sigma = s_1 s_2 \ldots s_{l-1} s_l$, a probabilistic suffix tree can be constructed in a similar way as constructing an ordinary suffix tree with two exceptions: (1) the tree is built on the reversed sequence $s_l \ldots s_1$; and (2) a probability vector is maintained for each node. The general process involves adding all suffixes (of $reverse(\sigma)$) to an empty tree[9] successively. The insertion of a suffix may involve updating counters associated with existing nodes, creating new nodes, and compute the probability vectors. Many algorithms [1, 5, 9, 11, 23, 28] have been proposed to efficiently build a suffix tree from a sequence. The computational complexity of building a suffix tree is linearly proportional to the length of the sequence. Due to space limitations, we will not elaborate on these algorithms. Interested readers please refer to individual papers for a detailed description. The probability vector associated with each node (whose label is σ') stores the (empirical) conditional probability distribution $P_S(s_i | \sigma')$ for $i = 1, \ldots, n$. The value of each entry $P_S(s_i | \sigma')$ is the ratio of the occurrence frequencies of two segments $\sigma' s_i$ and σ', i.e., $P_S(s_i | \sigma') = \frac{C_S(\sigma' s_i)}{C_S(\sigma')}$ where $C_S(x)$ is the count of the segment x. Note that the maintenance of probability vector of each node incurs little overhead since it can be updated incrementally every time the node is visited to increment the counter (when inserting a suffix).

Later, if a sequence has high similarity to this cluster and becomes a member of this cluster, the probabilistic suffix tree needs to be updated as well. Instead of using the entire sequence, only the segment that produces the highest similarity score to this cluster is used. All suffixes of the reverse of this

[9] An empty tree only contains the root node whose count is initialized to 0.

segment are inserted into the probabilistic suffix tree. The probability vector and the counter of relevant nodes are updated accordingly.

4.5 Cluster Consolidation

It is possible that some clusters are heavily overlapped, perhaps due to a bad initial choice of cluster seeds, especially when multiple sequences (actually) belonging to the same cluster are drawn to serve as the seeds. In such a case, we dismiss (small) clusters that are "covered" by others. This consolidating process examines each cluster sequentially in ascending order of their sizes, starting from the smallest one. For any cluster, if the number of sequences in this cluster which are not members of other (larger) clusters is very small (say, $< c$), then this cluster will not be retained. At the end of this consolidation process, all remaining clusters have substantial difference from each other.

4.6 Adjustment of the Similarity Threshold t

The initial value of t may not be the optimal one and therefore it is necessary to adjust the value of t during the clustering process. Here, we propose some simple but effective heuristic to determine the proper value of t. During each iteration of the clustering, a histogram is built to track the distribution of similarities of all sequence-cluster combinations[10] (Figure 3). In general, a huge number of sequence-cluster combinations would have low similarities and this number decreases as the similarity increases. Moreover, the decline in the number of sequence-cluster combinations usually does not following a straight line. The valley of the histogram curve (as shown in Figure 3) is a similarity (point) where the count declines much faster at the left than that at the right. Intuitively, the valley is the point where the histogram line makes the "sharpest turn". At any point on the histogram curve, the sharpness of the turn (at this point) can be measured by the angle between the regression lines of the left hand portion of the histogram curve and the right hand portion of the histogram curve, respectively. This angle is a monotonically decreasing function of the difference between the slopes of these two regression lines. The larger the slope difference, the smaller the angle and the sharper the turn. The objective thus becomes finding the point with the largest regression line slope difference.

Assume that the granularity of the histogram is $\frac{1}{n}$ of the domain. The histogram can be represented by a list of n points (x_i, y_i) $(i = 1, \ldots, n)$, where x_i is the median value of the similarity range corresponding to the ith bucket of the histogram and y_i is the number of sequence-cluster combinations whose similarity falling in the ith bucket. Then, the slope b_i^l of the regression line ($y = a_i^l + b_i^l x$) of the left hand portion is [15]

$$b_i^l = \frac{\sum_{j=1}^{i} x_j y_j - i \times \frac{\sum_{j=1}^{i} x_j}{i} \times \frac{\sum_{j=1}^{i} y_j}{i}}{\sum_{j=1}^{i} x_j^2 - i \times \left(\frac{\sum_{j=1}^{i} x_j}{i}\right)^2}.$$

Similarly, the slope b_i^r of the regression line ($y = a_i^r + b_i^r x$) of the right hand portion is

$$b_i^r = \frac{\sum_{j=i}^{n} x_j y_j - (n-i+1) \times \frac{\sum_{j=i}^{n} x_j}{n-i+1} \times \frac{\sum_{j=i}^{n} y_j}{n-i+1}}{\sum_{j=i}^{n} x_j^2 - (n-i+1) \times \left(\frac{\sum_{j=i}^{n} x_j}{n-i+1}\right)^2}.$$

The valley $\hat{t}$ is thus the point which maximize the difference between these two slopes.

$$\hat{t} = \max_{i=2}^{n-1} |b_i^l - b_i^r|$$

This process of finding the valley is linearly proportional to n. We omit the detailed analysis due to space limitations.

$\hat{t}$ can be regarded as a reasonable choice for the similarity threshold t. However, the value of $\hat{t}$ may change from one iteration to the next because it is influenced by the current value of t and the presence of noises. Therefore, instead of setting $t = \hat{t}$, we choose to adjust t towards $\hat{t}$ at a more conservative pace, i.e., the new value of t is set to $\frac{t+\hat{t}}{2}$. Note that if $\hat{t}$ remains stable, t will approach $\hat{t}$ very fast. In the case where the distance between t and $\hat{t}$ is small (e.g., $< 1\%$), we consider that they are virtually the same and will not change t any more.

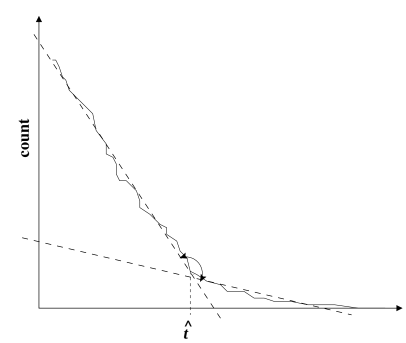

Figure 3. The Similarity Distribution

4.7 Complexity Analysis

The overall computational time greatly depends on the number of iterations actually executed. The computational complexity of each iteration is $O(N \times (k'l^2 + l^2)) =$

[10] Note that these similarities need to be calculated anyway during the sequence reclustering step for the purpose of seeking a better clustering.

$O(N \times k' \times l^2)$ where N, k', and l are the number of sequences in the database, the number of clusters, and the average sequence length, respectively. If the number of iterations used is M, then the overall computational complexity is $O(k'^2 l^2 + M \times N \times k' \times l^2) = O(M \times N \times k' \times l^2)$ since $k' \ll N$.

5 Discussions

In this section, we discuss some issues that influence the accuracy and efficiency of CLUSEQ. Due to space limitations, we only address the following two aspects.

5.1 Limited Memory Space

In practice, the memory space is usually limited and the size of each probabilistic suffix tree is also restricted by the available memory. Even though the significant nodes play a decisive role in similarity estimation, both significant and insignificant nodes are kept until the size of the tree reaches the memory limit. This is because an insignificant node in a tree may turn into a significant one if more sequences join the corresponding cluster. Once the size of a tree grows beyond this limit, some nodes have to be pruned. Several strategies can be employed to conduct the node pruning so that the remaining portion of the tree keeps as much information as possible.

1. *Prune node with smallest count first.* Intuitively, node with smaller count would have less chance to become significant node. Therefore, the pruning of such node(s) will be less likely to impact the accuracy of similarity estimation.

2. *Prune node with longest label first.* This is inspired by the *short memory property* that exhibits in many applications, which implies that the pruning of a node with longer label is expected to have less impact to the similarity estimation (than the pruning of a node with shorter label).

3. *Prune node with* **expected** *probability vector first.* This strategy only applies to the scenario where all insignificant nodes have been pruned already. Consider two significant nodes x and x' where x is a child of x' in the probabilistic suffix tree. Let σ and σ' be the labels of x and x', respectively. By definition, σ' is a suffix of σ. The probability vector $P(s_i|\sigma)$ (of node x) is considered as *expected* if it does not differ substantially from the probability vector $P(s_i|\sigma')$ (of node x'). If the node x is pruned, the node x' will be used as the substitute in the future similarity estimation. The less the difference between $P(s_i|\sigma)$ and $P(s_i|\sigma')$, the more accurate the estimated similarity.

With the above three strategies, little degradation of the accuracy of the similarity estimation can be observed in practice, even though a large number of nodes are pruned. This is further confirmed by the experimental results in Section 6.

5.2 Adjusted Probability Estimation

Since a probabilistic suffix tree captures empirical probability distribution on the next symbol given the preceding segments in a set of sequences. It is possible that, for some given segment $s_1 s_2 \dots s_{i-1}$, some symbol s_i has never been observed right after $s_1 s_2 \dots s_{i-1}$ in the sequences. That is, $P(s_i|s_1 s_2 \dots s_{i-1}) = 0$ empirically. This happens frequently to a small cluster and sometime creates an undesirable scenario. Consider the estimation of the probability of generating the sequence $\sigma = s_1 s_2 \dots s_l$. We have $P(\sigma) = P(s_1) \times P(s_2|s_1) \times \dots \times P(s_l|s_1 \dots s_{l-1})$. If $P(s_i|s_1 \dots s_{i-1}) = 0$, then $P(\sigma) = 0$ no matter how high the remaining conditional probabilities are. One way to solve this problem is to use the **adjusted** probabilities instead of the raw empirical probabilities in the computation so that no symbol will have an absolutely zero probability to appear no matter what segment is observed before it. Let p_{min} be the minimum probability that a symbol is observed after any segment. The adjustment is achieved by decreasing each nonzero empirical probability by a certain percentage so that a total of $n \times p_{min}$ is collected to be shared by all symbols where n is the number of distinct symbols. Thus, the adjusted probability of each entry $P(s_i|s_1 s_2 \dots s_{i-1})$ in the probability vector is $\hat{P}(s_i|s_1 s_2 \dots s_{i-1}) = (1 - n \times p_{min}) \times P(s_i|s_1 s_2 \dots s_{i-1}) + p_{min}$. Notice that this adjustment can be performed on the fly during the similarity estimation.

6 Experimental Results

We implemented the CLUSEQ algorithm in C programming language. All experiments were run on a Sun Ultrasparc 10 machine with 512 MB main memory and a 300 MHz CPU. In this section, we focus on three aspects of our algorithm: accuracy, scalability and sensitivity.

6.1 Accuracy of CLUSEQ Model

We utilize two real sequence databases to examine the accuracy of the CLUSEQ model. The first real data set we applied the CLUSEQ algorithm is a database of 8000 protein sequences [2]. These protein sequences belong to 30 different biological families. The size of each family ranges from 140 to 900. We compare the performance of our CLUSEQ model with three alternatives: edit distance (ED), edit distance with block operation (EDBO), hidden Markov model (HMM), and the q-gram based approach. The parameters of CLUSEQ is set to $k = 10$, $c = 30$, and $t = 1.0005$. Here we intentionally use the incorrect number of clusters and similarity threshold as the initial setting. To our delight, CLUSEQ is able to produce 30 clusters with high accuracy. The final value of t is 1.52 which is different from the initial setting. The number of states for HMM is 30 while q is set to 3 for the q-gram based approach. The result is reported in Table 2. The CLUSEQ model is clearly the winner among these five

Table 2. Model Comparison

Model	CLUSEQ	ED	EDBO	HMM	q-gram
Percentage of Correctly Labeled Proteins	82%	23%	80%	81%	75%
Response Time (sec.)	144	487	13754	3117	132

Table 4. Results from CLUSEQ

	English	Chinese	Japanese
Precision %	86	79	81
Recall %	84	78	80

models. The edit distance model produce clusters of very poor quality as a result of seeking global alignment but ignoring local alignments. Even though both the edit distance with block operation and the hidden Markov model manage to deliver comparable accuracy (above 80%), they both require expensive computation which causes a lengthy response time. In the q-gram based approach, each sequence is viewed as a set of segments of length q. The correlations among these q-grams are lost. As a result, although the q-gram based approach is relatively fast, its accuracy is much lower than the CLUSEQ model.

We now take a closer look of the results produced by CLUSEQ. Table 3 shows the precision and recall of ten families (among a total of 30 families) due to space limitations. The measures of precision and recall are defined as follows. For each protein family, let F be the set of proteins actually belong to the family and F' be the set of proteins assigned to this family by CLUSEQ. The precision is the ratio $\frac{|F \cap F'|}{|F'|}$ whereas the recall is $\frac{|F \cap F'|}{|F|}$. It is easy to see that the CLUSEQ algorithm performs well consistently for clusters of diverse sizes.

To demonstrate the wide applicability of CLUSEQ, we use a database of alphabet sentences in three natural languages: English, Chinese, and Japanese. For each language, we choose 600 sentences from the CNN web site (www.cnn.com), the Chinese news media Sina (www.sina.com.cn), and the Yahoo Japan web site (news.yahoo.co.jp), respectively. Sentences in both Chinese and Japanese are translated to the phonetic alphabet that utilizes the same set of alphabet as English. The space character is eliminated to create extra challenges. To introduce some noises, 100 sentences in other languages (e.g., Russian, German, etc.) are drawn randomly. Table 4 shows the statistics of the clustering results of CLUSEQ. The result of CLUSEQ is very promising. Among these three languages, English has the highest recall and precision because the English has some very distinct features, e.g., *"t" followed by "h"*, *"e" has a very high occurrence frequency*, etc. which is not present in the other two languages. Since these distinct features are easy to detect and commonly present in many English sentences, the precision and recall of English sentences is the highest. An interesting observation is that the majority of mislabelled English sentences are labelled as Chinese due to the fact that some prefixes and suffixes commonly occur in both languages, e.g., *ion*, *ch*, and *sh*. The Japanese has slightly worse precision and recall but is better than the Chinese. The most dominant rule in Japanese is the following: *a vowel is likely followed by a consonant and vice versa*. However, this rule is somewhat more difficult to detect since our algorithm does not integrate any *a priori* knowledge of consonants and vowels. The low precision and recall of the Chinese is due to the complicated features of the Chinese, which may be easily confused with other languages and cause the most of the mislabelling.

We also study the robustness of the CLUSEQ algorithm with respect to the number of outliers. In our experiments, the percentage of outliers varies from 1% to 20%. We find that the accuracy of CLUSEQ is immune to the increase of outliers.

6.2 Probabilistic Suffix Tree Size

Here, we study the effects of PST size on the performance of our CLUSEQ algorithm. In this experiment, we utilize a synthetic data set that consists of 100,000 sequences, each of which consists of 1000 symbols on average. There are 100 distinct symbols and we embed 50 clusters. Figure 4(a) and (b) shows the precision, recall and the response time of the clustering algorithm with respect to the maximum memory allocated for each tree. Based on Figure 4(a), we found that the improvement of the accuracy (i.e., recall and precision) is rather small when the allocated memory for a PST is at least 5MB. The response time continues to grow with larger PST size. As a result, we set the maximum PST size to be 5MB throughout the remainder of this section.

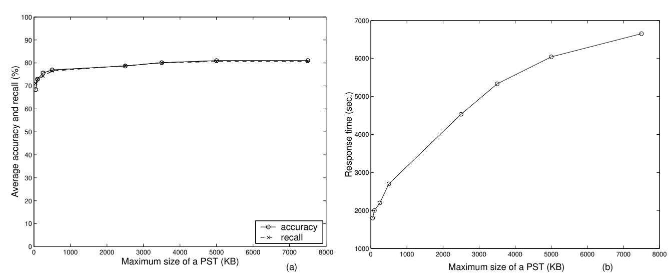

Figure 4. The effects of PST Size

6.3 Sensitivity Analysis

To understand the sensitivity of the CLUSEQ algorithm to the sample size m, we experiment with various m on a

Table 3. Results for CLUSEQ on Protein Database

Family	ig	pkinase	globin	7tm_1	homeobox	efhand	RuBisCO_large	...	gluts	actin	rrm
Size	884	725	681	515	383	320	311	...	144	142	141
Precision %	85	77	88	82	84	80	85	...	85	87	75
Recall %	82	89	86	83	81	83	80	...	89	85	82

data set with 100,000 sequences. There are 100 distinct symbols. Each sequence consists of 1000 symbols on average and there are totally 50 clusters and about 5% outliers. Figure 5 shows the effect of the initial sample size m to the quality and response time of the CLUSEQ algorithm. The results confirm with our previous discussion. The quality (i.e., precision and recall) of the CLUSEQ algorithm improves with the increase of m due to the fact that better initial clustering can be obtained. The improvement slows down when $m > 5 \times k$ where k is the number of clusters. On then other hand, the response time of the CLUSEQ falls into a valley as shown in Figure 5(b). When $m > 3 \times k$, the response time grows along with the increase of m in general, while the response time presents an opposite trend $m \leq 3 \times k$. After further investigation, we found that, with a small sample size, the quality of the initial clusters is very poor and it takes a longer course for CLUSEQ to reach the final clustering.

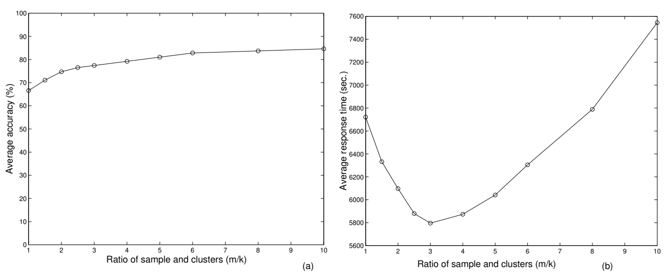

Figure 5. The effects of the number of initial samples

The CLUSEQ approach adjusts the cluster number k' and similarity threshold t dynamically. It does not require the user to set the optimal values for these two parameters. How fast and accurate CLUSEQ is able to approximate the true value of t and k is essential to the CLUSEQ approach. In this experiment, we embed 100 clusters in a data set of 100,000 sequences and 10% sequences are outliers. The similarity between each pair of sequences in the same cluster is at least 2 and the similarity between sequences from different clusters is always lower than 2 (i.e., the true value of t is 2). First we analyze the effects of variant initial cluster number k. Table 5 shows the final number of clusters and the response time for clustering as a function of the initial k value. It is clear that the CLUSEQ algorithm is able to approach the right number of clusters independent of the initial number of clusters. In addition, when the initial number of clusters is well of the actual number of clusters (e.g., two orders of magnitude less than the right value in our experiments), it takes about 60% more time to finalize the clustering process.

Table 5. Effects of initial number of clusters

init. cluster num.	1	20	100	200
final cluster num.	102	99	101	102
response time (sec)	10112	9023	6754	8976
precision	81.3%	82.1%	82.6%	81.0%
recall	81.6%	82.0%	83.4%	81.7%

Table 6. Effects of initial similarity threshold

init. t	1.05	1.5	2	3
final t	1.99	2.01	2.00	1.99
response time (sec)	8011	7556	6754	7234
precision	81.3%	83.1%	83.4%	81.9%
recall	82.1%	82.8%	83.6%	82.7%

We now study the parameter t. Table 6 shows the final value of t and the response time to finalize the clustering with respect to the initial value of t. We set k to 100 in order to isolate the effect. From the table, we can see that the final value of t is very close to the true value of t, i.e., 2 regardless of its initial setting. In addition, the extra cost incurred by setting a sub-optimal initial t is less severe, e.g., about 30% for using 1.05 as the initial t. In both experiments, we can see that CLUSEQ is very robust in the face of erroneous initial parameter setting.

Finally, we analyze the effects of the order to examine every sequence during each iteration of CLUSEQ. We compare the results of three orders. (1) Fixed order: sequences are processed according to their IDs. It means that the process order is the same for each round. (2) Random order: a new random permutation of sequences is used at each iteration. (3) Cluster-based order: all sequences assigned to the same cluster (in last iteration) are examined successively before the algorithm switches to sequences in another cluster. Both the fixed order and the random order are able to achieve reasonably high accuracy (82% and 83% respectively), while the cluster-based order delivers poor result (65% accuracy). Under a thorough investigation, we observe that the cluster-based order impairs the algorithm's ability to break the barrier of local optimum. The fixed order was used throughout the experiments because the random order may incur large disk I/Os.

6.4 Scalability of CLUSEQ Algorithm

There are four aspects of the scalability, with respect to the number clusters, number of sequences, average length of a sequence, and the number of distinct symbols. In this section, we analyze them one by one. We choose the initial sample size to be $5 \times k$ where k is the number of clusters, and the maximum size of a probabilistic suffix tree to be 5MB. In the following experiments, we assume there are 5% outliers in each data set.

To evaluate the scalability with respect to the number of clusters, four data sets are generated, each of which consists of 100,000 sequences. There are 100 distinct symbols and each sequence consists of 1000 symbols on average. We embed 10, 20, 50, and 100 clusters in each data set. Sequences in a cluster are all generated according to the same probabilistic suffix tree. Figure 6(a) shows the response time of CLUSEQ with respect to the number of clusters. From the figure, we can see that the response time is linearly proportional to the number of clusters, which coincides with the complexity analysis in Section 4.

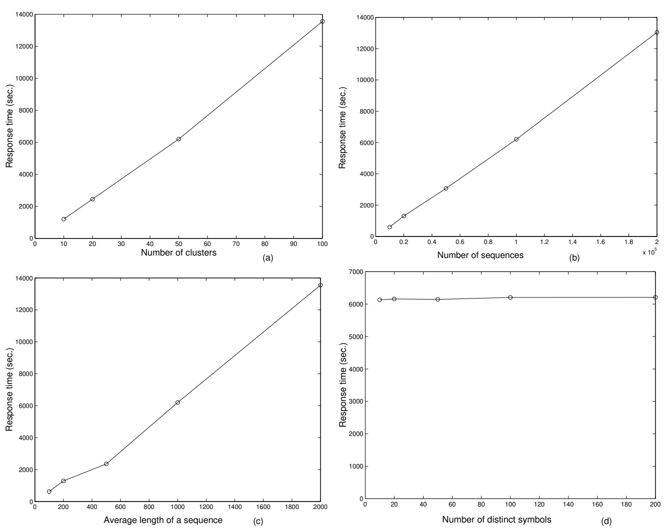

Figure 6. The scalability of CLUSEQ

Figure 6(b) shows the response time of the CLUSEQ with respect to the number of sequences. In this experiment, five data sets are employed, which consist of 10000, 20000, 50000, 100000, and 200000 sequences, respectively. There are 100 distinct symbols, and 50 clusters for each data set. On average, each sequence consists of 1000 symbols. From this figure, we can see that the response time also scales linearly with respect to the number of sequences. This result also agrees with the complexity analysis in Section 4.

Now we are analyzing the effect of average sequence length to the performance CLUSEQ. In this experiment, we use 100,000 sequences with five average lengths: 100, 200, 500, 1000, and 2000. We embedded 50 clusters in each data set. The total number of distinct symbols is 100. Figure 6(c) shows the response time as a function of the average sequence length. Even though the response time of the CLUSEQ algorithm is super-linear with respect to the length of sequences, the slope is very moderate as shown in Figure 6(c).

Finally, we study the effect of the number of distinct symbols to the performance of the CLUSEQ algorithm. Five data sets are used in this experiment, each of which consists of 100,000 sequences, each of which consists of 1000 symbols. There are 50 clusters embedded in each data set. The data sets differ on the number of distinct symbols. From Figure 6(d), we can see that the number of distinct symbols has little impact on the response time.

7 Related Work

In this section, we give a brief overview of some recent research advances that are closely related to our work presented in this paper.

7.1 Applications of Suffix Tree

The suffix tree structure has been used extensively in many areas besides its original indexing function in substring matching. It has been proven to be a very successful model to capture significant patterns in sequences. The substring selectivity estimation in a text database [13, 14, 16] is one of the successful achievements. The objective is to obtain a good estimate for a given substring matching query. A (pruned) suffix tree is built for the entire database where each node is associated with a counter to record the number of occurrences of the corresponding substring. The selectivity of a given substring is inferred from relevant counters under some statistical assumptions. Another application of suffix tree is to accelerate the protein classification [3, 4]. The suffix tree is constructed to organize significant protein regions to facilitate the detection of highly conserved protein regions according to some scoring matrices. It has been demonstrated that the application of suffix tree not only can produce dramatic performance improvement, but also can incorporate biological considerations such as amino acid background probabilities and amino acid substitution probabilities.

7.2 Clustering in Categorical Domain

Clustering in categorical domain [6, 7, 10, 20] has been studied recently and the research that is most related to our work is along two themes: *document clustering* and *transaction clustering*. Document clustering [12, 17, 24, 25, 29] has long been an important problem in the information retrieval field. Each document is viewed as a set of words and each word may appear in multiple documents. A document cluster is defined as a set of documents that are similar to each other. Similarly, the transaction clustering [10] studied in the data mining field takes a transaction database (e.g., market basket database) as the input, where each transaction contains a set of items. Despite the differences in problem formation and similarity definition, all of the above work considered non-sequential domain and disregard the order of words/items in a document/transaction. It is worth mentioning that, even though the suffix tree was used to store identified phrases in [29], each document is still treated as a collection of isolated phrases that are assumed totally independent from each other. In contrast, we consider the problem of clustering sequential data and focus on exploring the *sequential* characteristics of the data in this paper.

8 Conclusions

In this paper, we investigated in the problem of clustering sequences. A novel model (CLUSEQ) is proposed for sequence cluster by exploring significant statistical properties possessed by the sequences. The conditional probability distribution (CPD) of the next symbol given a preceding segment is derived and used to characterize sequence behavior and to support the similarity measure. A new variation of the suffix tree, namely *probabilistic suffix tree*, is employed to organize (the significant portion of) the CPD in a concise way. A novel algorithm is devised to efficiently discover clusters with high quality and is able to automatically adjust the number of clusters to its optimal range via a unique combination of successive new cluster generation and cluster consolidation. The performance of CLUSEQ has been demonstrated via extensive experiments on several real and synthetic sequence databases.

References

[1] A. Apostolico and G. Bejerano. Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. *Proc. of ACM RECOMB*, pp. 25-32, 2000.

[2] The SWISS-PROT protein sequence data bank. *Nuci. Acids Res.*.

[3] G. Bejerano and G. Yona. Modeling protein families using probabilistic suffix trees. *Proc. of ACM RECOMB*, pp. 15-24, 1999.

[4] B. Dorohonceanu and C. Nevill-Manning. Accelerating protein classification using suffix trees. *Proc. of Intelligent Systems for Molecular Biology*, 2000.

[5] M. Farach, P. Ferragina, and S. Muthukrishnan. Overcoming the memory bottleneck in suffix tree construction. *Proc. of IEEE FOCS*, pp. 174-183, 1998.

[6] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS–clustering categorical data using summaries. *Proc. of ACM SIGKDD*, pp. 73-83, 1999.

[7] D. Gibson, J. M. Kleinberg, P. Raghavan. Clustering categorical data: an approach based on dynamical systems. *Proc. 24th VLDB*, pp. 311-322, 1998.

[8] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. *Proc. 27th VLDB*, pp. 491-500, 2001.

[9] R. Grossi and J. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *Proc. of ACM STOC*, pp. 397-406, 2000.

[10] S. Guha, R. Rastogi, and K. Shim. ROCK: a robust clustering algorithm for categorical attributes. *Proc. of ICDE*, pp. 512-521, 1999.

[11] D. Gusfield. Algorithms on strings, trees, and sequences. *Cambridge University Press*, 1997.

[12] V. Hatzivassiloglou, and L. Gravano, and A. Maganti. An investigation of linguistic features and clustering algorithms for topical document clustering. *Proc. of ACM SIGIR*, 2000.

[13] H. Jagadish, O. Kapitskaia, R. Ng, and D. Srivastava. Multi-dimensional substring selectivity estimation. *Proc. of VLDB*, 1999.

[14] H. Jagadish, R. Ng, and D. Srivastava. Substring selectivity estimation. *Proc. of ACM PODS*, 1999.

[15] R. Jain. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, Inc., 1991.

[16] P. Krishnan, J. Vitter, and B. Iyer. Estimating alphanumeric selectivity in the presence of wildcards. *Proc. of ACM SIGMOD*, pp. 282-293, 1996.

[17] R. Krishnapuram, A. Joshi, O. Nasraoui, and L. Yi. Low-complexity fuzzy relational clustering algorithms for web mining. *IEEE Trans. Fuzzy Systems*, 2001.

[18] J. Lin. Divergence measures based on the Shannon entropy. *IEEE Tran. on Information Theory*, vol. 37, no. 1, pp. 145-151, 1991.

[19] D. Loprestli and A. Tomkins. Block edit models for approximate string matching. *Theoretical Computer Science*, 1996.

[20] T. Morzy, M. Wojciechowski, and M. Zakrzewicz. Scalable hierarchical clustering method for sequences of categorical values. *Proc. PAKDD*, pp. 282-293, 2001.

[21] S. Muthukrishnan and S. Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. *Proc. of ACM Symposium on Theory of Computing*, pp. 416-422, 2000.

[22] G. Navarro and R. Baeza-Yates. A practical q-gram index for text retrieval allowing errors. *CLEI Electronic Journal*, vol 1(2), 1998.

[23] D. Ron, Y. Singer, N. Tishby. The power of anmesia: learning probabilistic automata with variable memory length. *Machine Learning*, vol. 25, no. 2-3, pp. 117-149, 1996.

[24] N. Slonum and N. Tishby. Document clustering using word clusters via the information bottleneck method. *Proc. of ACM SIGIR*, 2000.

[25] N. Slonim and N. Tishby. The power of word clusters for text classification. *Proc. of ECIR*, 2001.

[26] E. Sutinen and . Tarhio. Filtration with q-samples in approximate string matching. *Proc. 7th Annual Symp. Combinatorial Pattern Matching*, pp. 50-61, 1996.

[27] E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, vol 92(1), pp. 191-202, 1992.

[28] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, vol 14, pp. 249-260, 1995.

[29] O. Zamir and O. Etzioni. Web document clustering: a feasibility demonstration. *Proc. ACM SIGIR*, 1998.