Towards Automatic Clustering of Protein Sequences

Jiong Yang
T.J. Watson Research Center
IBM
jiyang@us.ibm.com

Wei Wang
Department of Computer Science
University of North Carolina, Chapel Hill
weiwang@cs.unc.edu

Abstract

Analyzing protein sequence data becomes increasingly important recently. Most previous work on this area has mainly focused on building classification models. In this paper, we investigate in the problem of automatic clustering of unlabeled protein sequences. As a widely recognized technique in statistics and computer science, clustering has been proven very useful in detecting unknown object categories and revealing hidden correlations among objects. One difficulty that prevents clustering from being performed directly on protein sequence is the lack of an effective similarity measure that can be computed efficiently. Therefore, we propose a novel model for protein sequence cluster by exploring significant statistical properties possessed by the sequences. The concept of imprecise probabilities are introduced to the original probabilistic suffix tree to monitor the convergence of the empirical measurement and to guide the clustering process. It has been demonstrated that the proposed method can successfully discover meaningful families without the necessity of learning models of different families from pre-labeled "training data".

1 Introduction

During recent years, many efforts have been carried out on analyzing protein sequences. Research (Apostolico&Bejerano, 2000, Bailey&Grundy, 1999, Bejerano&Yona, 1999, Dorohonceanu&Nevill-Manning, 2000) in this area focuses on modeling and classifying protein families based on function, structure, and homology. Despite the differences in model definition and/or algorithm details, a major assumption of previous work is that the protein families or functional categories are known in advance and the protein sequences used to build the classification model are properly labeled with the corresponding families/categories. In this paper, we are studying a slightly different but more challenging problem — clustering unlabeled protein sequences into groups/families/categories automatically.

Clustering has been widely recognized as a powerful technique in statistics and computer science, 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. Most research on clustering focused on numerical domains and assumed metric space in many cases. In many of these models, the similarity between objects can be defined in a clear and relatively straightforward way. In most of previous work, the similarity between two protein sequences is measured by the maximum alignment between them, e.g., ProtoMap (http://www.protmap.cs.huji.ac.il). To employ the maximum alignment method, the similarity between each pair of sequences is computed and based on the pair-wise similarity, a directed graph is constructed with the vertices representing the sequence and the edges representing the similarity. Finally, the clusters of sequences are the strongly connected components in the graph. However, this approach suffers from two aspects. (1) The computation is costly. To compute the pairwise similarity of each sequence, the complexity is at least $O(N^2 \times l^2)$ where N and l are the number of sequences and the average length of a sequence in the database, respectively. The second step of finding the strongly connected component may also be time consuming depending on the actual method used. (2) The incremental clustering would be very inefficient. After clustering a set of unlabeled sequences, if more sequences need to be clustered, then all the pair-wise similarity between the unclustered and clustered sequences, and among the unclustered sequences need to be computed. The complexity is at least $O(N \times M \times l^2)$ where M is the number of new sequences. In reality, the protein sequences are not available at once. Almost there are new sequences available at each month. As a result, it is beneficial to employ a method which combines the model and clustering of protein sequence together so that once a new sequence needs to be clustered, we can simply compare to the discovered models of the clusters to identify which cluster it should be in or it belongs to a new cluster (if it does not fit to any model).

An alternative approach is to utilize the suffix tree structure to explore significant patterns of the protein sequences and use these patterns to evaluate the similarity, originally introduced in Bejerano&Yona(1999) and Ron et al.(1996). More specifically, these patterns can be used to assess the similarity between a pair of protein sequences and between a protein sequence and a cluster (of protein sequences). Under this model, protein sequences belonging to one cluster may subsume to the same (or a similar) probability distribution of amino acids (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 (Bejerano & Yona, 1999). The suffix tree (or its variations) (Farach, 1997, Grossi&Vitter, 2000, Gusfield, 1997, McCreight, 1976, Ron et al., 1996, Ukkonen, 1995) has been proven to be a very successful model to capture significant patterns in symbol sequences. With regard to the problem addressed in this paper, the suffix tree structure can be utilized to serve as a compact representation to summarize significant patterns shared by members of each cluster. The model can be easily used to identify and organize significant appearances of segments among a cluster of sequences, regardless of the relative positions of these segments within different sequences. These significant patterns imply certain features that is common to a large subset of sequences in the cluster and reveal important statistical properties of the cluster. In practice, they induce a (conditional) probability distribution of the next symbol given the preceding segments, which can be used in estimating the similarity between a sequence and a cluster. The key idea is that, 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.

In this paper, we also adopt the suffix tree structure as the foundation to support our similarity measure. In addition, we introduce a novel technique to determine the convergence of each cluster and to incorporate available prior knowledge. A pair of upper and lower probabilities is used to quantify the range of the true probability. It has been proven that such probability range is guaranteed to converge to a single value after sufficient trials if the underlying probability distribution remains static. An open gap between the upper and lower probabilities (after sufficient trials) is indeed a good indication of the heterogeneity of the probability distributions, which in turn implies that the corresponding sequences belong to multiple clusters. Therefore, the convergence of the imprecise probabilities can well serve as the touchstone to trigger necessary cluster split and to determine the termination of the clustering process. This provides the opportunity to design an integrated clustering process that allows a more flexible and systematic switch between model training and splitting, comparing to the annealing method proposed in Bejerano et al. (2001) which consists of iterations of soft clustering and progressive refinement.

As we shall explain in more detail later, a suffix tree is maintained for each (potential) cluster of protein sequences in our proposed scheme. Each node x in the suffix tree is associated with an occurrence count $count(x)$ and a vector of n entries, each of which corresponds to a distinct symbol, d, and consists of an empirical probability $prob(d)$ and a pair of lower and upper probabilities $(\underline{prob}(d), \overline{prob}(d))$. The counter $count(x)$ keeps track of the number of occurrences of the path label $\sigma(x)$ of x. The probability vector monitors both the empirical value and the potential range of the conditional probability of observing the symbol, say d, right after $\sigma(x)$ in a protein sequence. Starting from $[0, 1]$ or some other initial range based on the prior knowledge, the range defined by the lower and upper probabilities will be continuously refined and eventually converge during the clustering process.

The remainder of this paper is organized as follows. Section 2 gives some background knowledge of the imprecise probabilities. The formal definition of the protein sequence cluster is provided in Section 3. Section 4 presents the clustering algorithm while Section 5 discusses some implementation issue. Some experimental results are shown in Section 6. The conclusions are drawn in Section 7.

2 Background on Imprecise Probability

The concept of imprecise probability was introduced in Walley (1991) to accommodate uncertainty due to the lack of knowledge or information conflict. In contrast to the traditional probability measure that utilizes a single number to represent the likelihood of an event, the imprecise probability measure employs a pair of numbers, namely a **lower probability** and an **upper probability**, to represent the range of the likelihood of an event. This provides the opportunity to represent the degree of uncertainty in addition to the probability expectation. The difference between the upper probability and the lower probability is defined as the **degree of imprecision**. For instance, the probability (of traditional meaning) of the event that a "head" occurs from a toss of a coin is 0.5 if no information is available on the fairness of the coin while the imprecise probability of the event is $[0, 1]$ and the degree of imprecision is 1.

Let $\overline{P}(y)$ and $\underline{P}(y)$ be the (prior) upper and lower probabilities of the event y, the posterior upper and lower probabilities after observing a occurrences out of a set of b trials $(b \geq a)$ can be obtained as follows.

$$\overline{P'}(y) = \frac{\tau \times \overline{P}(y) + a}{\tau + b} \qquad \underline{P'}(y) = \frac{\tau \times \underline{P}(y) + a}{\tau + b}$$

τ is called the *learning parameter* and is used to control the weight of the prior probabilities in calculating the posterior probabilities. For example, if the prior probability and the current experiments (i.e., observing a occurrences out of a set of b trials) have equal weight towards the posterior probability, then we have $\tau = b$ and

$$\overline{P'}(y) = \frac{\overline{P}(y) + \frac{a}{b}}{2} \qquad \underline{P'}(y) = \frac{\underline{P}(y) + \frac{a}{b}}{2} \tag{1}$$

The higher the value of τ, the more the weight of the prior probabilities and the less the weight of the current experiments. It has been proven (Walley, 1991) that

1. if $\underline{P}(y) \leq \frac{a}{b} \leq \overline{P}(y)$ (that is, the result of the experiments is consistent with the prior probabilities), then the *posterior* degree of imprecision $\overline{P'}(y) - \underline{P'}(y)$ is less than or equal to the *prior* degree of imprecision $\overline{P}(y) - \underline{P}(y)$; and

2. $\lim_{b\to\infty} \overline{P'}(y) = \lim_{b\to\infty} \underline{P'}(y) = \lim_{b\to\infty} \frac{a}{b}$ regardless of the prior probabilities.

These two properties are very important and provide the motivation and justification of our cluster splitting strategy and guarantee the termination of our clustering algorithm in practice.

3 Protein 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. In the domain of protein sequences, there are totally 20 amino acids, each of which is represented by a distinct symbol. A **sequence** is an ordered list of symbols in $\Im$. The number of symbols in a sequence is referred to as the **length** of the sequence. Given a sequence, a **segment** is defined as a *consecutive* portion of the sequence. For example, “ba” is a substring of “abab” while “aa” 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 in this paper. 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. The similarity measure is built upon statistical properties of the sequences. 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 (e.g., the Kullback-Leibler Divergence (Lin, 1991)) to assess the difference between two probability distributions. The main theme 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, then there are $O(|\Im|^L)$ distinct segments. The computational complexity of calculating the difference between two probability distributions is exponential with respect to the length of the segment considered. 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. Looking ahead, the operation of difference calculation will be performed numerous times during the clustering process and contributes the majority of the entire execution time. To avoid the expensive distance computation, we employ an alternative method (Bejerano&Yona, 1999). 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

$$\begin{aligned} P_S(\sigma) &= P_S(s_1) \times P_S(s_2|s_1) \times P_S(s_3|s_1 s_2) \times \ldots \\ &\quad \times P_S(s_l|s_1 s_2 \ldots s_{l-1}) \\ &= \Pi_{i=1}^{l} P_S(s_i|s_1 \ldots s_{i-1}) \end{aligned}$$

where $P_S(s_i|s_1 s_2 \ldots s_{i-1})$ 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. This predict probability can serve as an indicator of the similarity between the sequence σ and the cluster S. If the value of $P_S(\sigma)$ is sufficiently high, then we may conclude that the sequence σ subsumes a similar CPD to that of S and may be considered a member of S. To accommodate sequences of different lengths, the value of $P_S(\sigma)$ should be normalized by the length of σ. The probability $P_r(\sigma)$ that σ is generated by a random generator according to the symbol distribution in the sequence database can be used as the basis of the normalization. We have $P_r(\sigma) = \Pi_{i=1}^{l} p(s_i)$ where $p(s_i)$ is the probability of observing the symbol s_i in any give position of any sequence in the entire database. The similarity between the sequence σ and the cluster S can then be defined as

$$\begin{aligned} sim_S(\sigma) &= \frac{P_S(\sigma)}{P_r(\sigma)} = \frac{\Pi_{i=1}^{l} P_S(s_i|s_1 \ldots s_{i-1})}{\Pi_{i=1}^{l} p(s_i)} \\ &= \Pi_{i=1}^{l} \left(\frac{P_S(s_i|s_1 \ldots s_{i-1})}{p(s_i)} \right). \end{aligned}$$

Note that this similarity measure works well under the assumption that the entire sequence of σ follows a single CPD.

This may not be always true. Sometimes different portions of a sequence may subsume to different CPDs. (An example would be the multi-domain proteins.) The above similarity needs to be modified to accommodate this situation. The **similarity** between a sequence σ and a cluster S is the *maximum* similarity between any continuous segment of σ and S. That is

$$Sim_S(\sigma) = \max_{1 \leq i \leq j \leq l} sim_S(s_i \dots s_j). \tag{2}$$

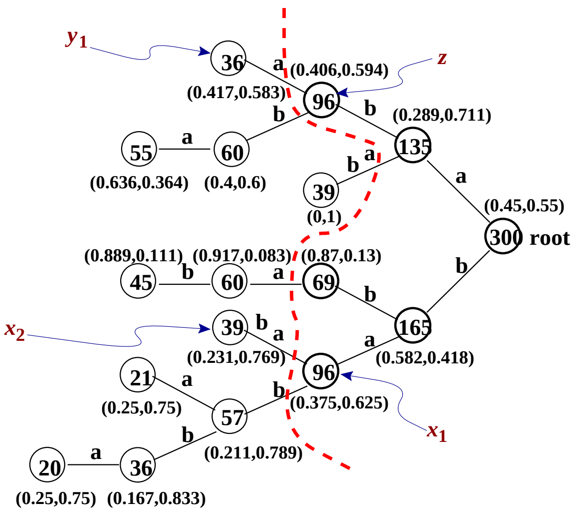
$Sim_S(\sigma) > 1$ indicates that there is some evidence to show that the sequence σ subsumes the CPD of S; and the higher the value of $Sim_S(\sigma)$, the stronger the evidence. If a sequence σ produces a small $Sim(\sigma)$ (e.g., less than 1) for every cluster, then σ is deemed to be an outlier. A threshold t ($t \geq 0$) is employed to separate clustered sequences from outliers. t is referred to as the **similarity threshold** in the remainder of this paper. If the value of $Sim(\sigma)$ 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 3.1 *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.*

The threshold t is utilized to control the cluster quality. 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 specified by the user due to its application dependent nature. An extensive study regarding this issue is left to the full version of this paper. Given a sequence database and a user specified parameter k, our objective is to group these sequences into k' clusters where $k' \geq k$. (The default value of k is 2.) Note that it is possible that a sequence belongs to multiple clusters since different portions of the sequence may deliver high similarity scores to different clusters. Before we formally present the clustering algorithm, we first give some terminology used in the following discussion and describe the data structure employed during the clustering process. 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 $i = 1, 2, \dots, l'$. 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 $i = 1, 2, \dots, l'$. 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 will be incurred. Given a sequence database and a pre-specified threshold c, a segment is called a **significant segment** if it appears at least c times in the database, and is called an **insignificant segment** otherwise. c is referred to as the **significance threshold**.

During the clustering process, a probabilistic suffix tree is utilized to store the "summary information" of each cluster. The probabilistic suffix tree (PST) was originally introduced in Ron et al. (1996) and used later in Bejerano&Yona (1999) for identifying and organizing significant segments among input sequences. A PST on a set of sequences over an alphabet is a rooted directed tree where the path from the root to each node corresponds to a distinct segment that appears in the sequence set. Figure 1(a) shows a portion of a PST in landscape mode. The concatenation of the edge-labels on the path *from a node to the root* exactly spells out a segment in the sequence set. This segment will be referred to as the **path label** of the node in the remainder of this paper. We sometimes also say a node is **labeled** with a segment σ' if σ' is the path label of the node. For example, ba is the path label of node z in Figure 1(a). A key feature of the PST is that each node is also associated with a probability distribution vector $(prob(s_1), prob(s_2), \dots, prob(s_n))$ over the alphabet $\Im = \{s_1, s_2, \dots, s_n\}$, which corresponds to the probability distribution of the next symbol given the path label of the node as the preceding segment. The tuple $(0.406, 0.594)$ at node z represents that the probability of observing a right after the segment ba and the probability of observing b after ba are 0.406 and 0.594, respectively. (That is, $prob(a|ba) = 0.406$ and $prob(b|ba) = 0.594$.) In addition, a counter may be associated with each node to track the number of occurrences of the corresponding path label. The number inside each node (e.g., 96 in node z) in Figure 1(a) is the count of the corresponding path label (e.g., ba) in the sequence set. A node is a **significant node** if its path label occurs at least c times in the sequence set and is an **insignificant node** otherwise. The dashed line separates the significant nodes from the rest if $c = 65$ in Figure 1(a).

Looking ahead, the clustering process terminates when the probability distribution vector associated with each *significant* node has been "stabilized". The imprecise probability comes into play in determining the convergence of each entry of the probability distribution vector. A pair of lower probability and upper probability is maintained for each entry during the course of clustering to quantify the potential variation of the actual probability. Therefore, each entry in the probability distribution vector now contains a triple $(prob(d), \underline{prob}(d), \overline{prob}(d))$ where $prob(d)$ is used to store the (empirical) probability of observing symbol d right after observing the path label of the node and the pair $[\underline{prob}(d), \overline{prob}(d)]$ represents the potential range where the true probability fails in, which is computed from previous experiments and/or prior knowledge. A probability entry is considered *stabilized* if the gap between the lower and upper probabilities diminishes. Given a pre-specified threshold ϵ, we said that a node **converges** if the difference between the pair of upper and lower probabilities is below ϵ for every entry in the probability distribution vector associated with this



(a) A Probabilistic Suffix Tree

	prob	$\underline{\text{prob}}$	$\overline{\text{prob}}$
a	0.406	0.3	0.5
b	0.594	0.55	0.6

(b) Probability Distribution Vector of Node z

Figure 1. The Probabilistic Suffix Tree

node. ϵ will be referred to as the **convergence threshold** in the remainder of this paper. We refer to the tree with the imprecise probabilities as the **imprecise probability suffix tree** (IPST) in order to distinguish it from the original probability suffix tree (PST). An IPST is said to **converge** if every *significant node* in the tree converges. Figure 1(b) shows an example of the probability distribution vector of node z in Figure 1(a) when the imprecise probabilities are employed. it is easy to see that node z does not converge at this moment if $\epsilon = 0.01$.

Given a segment σ' and a node x in the IPST, x is called the **prediction node** of σ'

1. if σ' is significant, and
   - σ' is the path label of x or
   - σ' is a proper *suffix* of x's path label and the path label of x's parent is a proper *suffix* of σ';
2. if σ' is insignificant and the path label of x is the *longest significant suffix*[1] of σ'.

For example, the prediction node of *ab* is x_1 while the prediction node of *abba* is z in Figure 1(a).

4 Algorithm

Our proposed clustering algorithm (CLUSEQ) uses an imprecise probability suffix tree to store significant features of each sequence cluster. The pseudo-code is presented in Algorithm A.1 (in the appendix). The CLUSEQ algorithm takes a sequence database Σ together with four parameters k, c, t, ϵ as the input and produces a set of $k'(k' \geq k)$ clusters. At the beginning, all sequences in the database are unclustered. The general idea of CLUSEQ is that, starting from a set of k initial clusters (Line 1), an iterative process (Line 4 to 43) is employed to continuously improve the quality of the clustering. Each initial cluster contains only one sequence. To obtain the k initial clusters, k sequences that have little similarity to each other are drawn from the database Σ. While an IPST is maintained for each cluster during the mining process, an IPST is constructed for each drawn sequence to represent the status of the corresponding initial cluster. Later, during each iteration, a sequence will be examined against the IPST of every cluster (Line 7 to 16) to identify the similar one(s) and to update the IPST(s) accordingly. Given a sequence σ and a cluster T_j, the similarity can be calculated according to Equation 2. A sequence is assigned to the cluster(s) that produces sufficiently high similarity. If σ has low similarity to every cluster, then it would remain "unclustered" (i.e., as an outlier). The similarity threshold t is employed (Line 11) to separate clustered sequences from the rest. The sequence σ can be considered a (new) member of some cluster only if the similarity is greater than t. Then, a validation process is performed (Line 20 to 38) to check the convergence of each cluster and whether a split is necessary. At the end of each iteration, we also check (Line 39 to 43) whether there exist a large number of outliers (e.g., more than 10% of overall population). If so, it is possible that some cluster is still missing. In this case, additional seed(s) will be generated from outliers to initiate new clusters. The entire clustering procedure ends when the every IPST converges. We now discuss each step in detail in the following subsections.

[1] Given a segment $s_1 s_2 \ldots s_i$, a suffix $s_j \ldots s_i$ is called the **longest significant suffix** if $s_j \ldots s_i$ is significant and any longer suffix $s_{j'} \ldots s_i$ is insignificant, where $1 \leq j' < j$.

4.1 Similarity Estimation

The similarity estimation is very crucial to both the accuracy and efficiency of the CLUSEQ algorithm. Given an imprecise probabilistic suffix tree $IPST$ that models the cluster S, the similarity of a sequence $\sigma = s_1 s_2 \ldots s_l$ to S is defined by Equation 2. A dynamic programming method can be used to calculate $Sim_S(\sigma)$ via a single scan of σ. Let

$$X_j = \frac{P_S(s_j | s_1 \ldots s_{j-1})}{p(s_j)},$$

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

$$Z_j = \max_{1 \leq i1 \leq i2 \leq j} sim_S(s_{i1} \ldots s_{i2}).$$

Then, $Sim_S(\sigma)$ can be obtained by

$$Y_j = \max\{Y_{j-1} \times X_j, X_j\},$$

$$Z_j = \max\{Z_{j-1}, Y_j\},$$

and $Sim_S(\sigma) = Z_l$.

The value of $P_S(s_i | s_1 s_2 \ldots s_{i-1})$ can be obtained by locating the prediction node of $s_1 s_2 \ldots s_{i-1}$ in $IPST$. This can be done by traversing from the root of $IPST$ along the path $s_{i-1} \rightarrow \ldots \rightarrow s_2 \rightarrow s_1$ until any further advance (along the path) would cause the count below the threshold c. For example, during the estimation of $P(b|bba)$, the prediction node of bba in the tree in Figure 1(a) can be located by traversing from the root along the path $a \rightarrow b \rightarrow b$. If $c = 65$, then we stop at node z ($pn = z$) in Figure 1(a), whose path label is $\sigma' = ba$. We can then obtain the value of $P_S(s_i | s_1 s_2 \ldots s_{i-1})$ by retrieving the entry corresponding to s_i in the probability vector associated with the prediction node pn. That is, $P_S(s_i | s_1 s_2 \ldots s_{i-1}) = P_S(s_i | \sigma') = pn.prob[s_i]$. In the above example, $P(b|bba) = P(b|ba) = z.prob[b] = 0.594$.

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.

4.2 Seed Generation

Let k be the number of clusters that need to be initiated at the beginning of the process. The goal of this step is to generate the seed clusters. Since each cluster is represented by an imprecise probabilistic suffix tree. The function $InitialSeed()$ (Line 1 of Algorithm A.1) returns a set of k IPSTs. Intuitively, each seed should have as little similarity to other seeds as possible. A straightforward way to generate seeds is to compute the pairwise similarity between every pair of sequences and then choose k sequences with least similarity to each other to serve as the k initial clusters. This would require $\Theta(|\Sigma|^2)$ similarity computation which is very inefficient when Σ is a large database. 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 sequences $S_1, S_2, \ldots, S_m$ are selected randomly from the sequence database Σ where $m \geq k$. For each sample sequence S_i, an IPST $IPST_i$ is constructed. From these m IPSTs, k trees will be chosen as the initial seeds. The following heuristics can be used to choose the optimal seeds. Let T be the set of seeds. T is initialized to be empty. A greedy algorithm is carried out, which consists of k steps. At each step, each remaining sample sequence is examined to calculate the highest similarity to any seed in T, and among them, the sequence with the least similarity to all seeds in T is selected and put in T. At the end of this procedure, T would consists of k seeds.

One question that may be raised at this point is how to set the value of m. A very large value of m would assure a promising quality of the generated seeds but incur significant computation, while a very small m would generate less optimal seeds, which in turn may cause the algorithm to consume longer time to reach the termination condition. (This can be observed from the experimental result in Section 6.) The best scenario is that a sequence in each of the k clusters is selected as a seed. To make it possible, at least one sequence in each cluster should be in the sample set. For the sake of simple explanation, let's assume that all clusters are of the same size and a randomly drawn sequence has $1/k$ probability to belong to each cluster. The probability that no sequence has been drawn from a given cluster after m trials is $\left(\frac{k-1}{k}\right)^m$. Then the probability that there exists a cluster from which no sequence has been drawn in the sample is less than $k \times \left(\frac{k-1}{k}\right)^m$. Thus the probability that at least one sequence is drawn from each cluster is *greater than* $1 - k \times \left(\frac{k-1}{k}\right)^m$. As we can see from the plot in Figure 2, this probability approaches 1 quickly with increasing value of m and the probability is sufficiently high when $m = 5k$. (The x-axis in Figure 2 represents the ratio $\frac{m}{k}$.) Note that the curves in Figure 2 are (conservative) lowerbounds of the actual probabilities. (The actual probabilities should be much higher.) In the remainder of this paper, we set $m = 5k$ unless otherwise specified. The computational complexity of the seed generation process is $O(m^2 \times l^2) = O(k^2 \times l^2)$ where k and l are the number of clusters and the average sequence length, respectively.

During the course of clustering, it may be necessary to initiate additional clusters. In this case, additional seed(s) (Line 42 in Algorithm A.1) can be generated in a similar fashion to the generation of initial seeds with one exception: the addi-

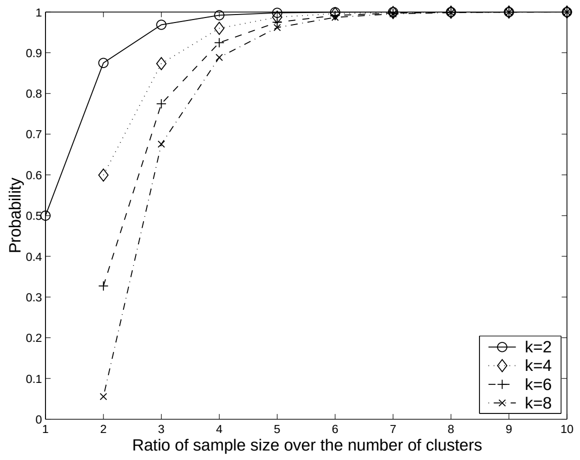


Figure 2. The Probability of Drawing at least one Sequence from each Cluster

tional seed(s) (chosen from the set of outliers) need to have low similarities not only to each other but also to existing clusters.

4.3 Imprecise Probabilistic Suffix Tree Construction for a Sequence

Given a sequence $\sigma = s_1 s_2 \ldots s_{l-1} s_l$, an imprecise probabilistic suffix tree can be constructed by in a similar way to build a probabilistic suffix tree (Bejerano&Yona, 1999). The only difference is that, in an IPST, each probability entry associates with a pair of lower and upper probability bounds. The values of these lower and upper probabilities are initialized to be 0 and 1 respectively. The computational complexity of building an imprecise probabilistic suffix tree is linearly proportional to the length of the sequence. Due to the space limitations, we will not elaborate on these algorithms. Interested readers please refer to the individual papers for a detailed description.

4.4 Updating the Imprecise Probabilistic Suffix Tree

Every time a sequence is considered similar to a cluster, the imprecise probabilistic suffix tree of the cluster should be updated accordingly (Line 13 in Algorithm A.1). Instead of inserting the entire sequence to the imprecise probabilistic suffix tree, only the portion that produces the high similarity scores will be used. This procedure is exactly the same as maintaining a probabilistic suffix tree since the imprecise probability ranges only need to be recalculated once for each iteration and can be performed (Line 26 to 27) during the cluster validation. The computational complexity of updating the probabilistic suffix tree(s) to reflect the change regarding a sequence σ is $O(l^2)$ where l is the length of σ.

4.5 Cluster Convergence and Split

A validation process (Line 20 to 38 in Algorithm A.1) is carried out to on each cluster to check whether the imprecise probabilistic suffix tree converges and whether a cluster split needs to be performed. The convergence of an IPST can be tested via a traversal of the IPST. For each node in the tree, the lower and upper probabilities of each symbol d is updated according to Equation 1 (Line 26, 27). Here, we assume that the prior probability and empirical probability of the current iteration have the same weight towards the posterior probability for the sake of simplicity. If the node is a significant node and these two probabilities differ more than ϵ for any symbol, then the node is considered *not converged* (Line 32 to 34). If any significant node does not converge, then the IPST is considered not converged. It is possible that an IPST will never converge. This may happen when the IPST indeed represents a mixture of multiple CPDs. To address this scenario, during the convergence test, if the average degree of imprecision[2] of significant nodes fails to improve from previous iteration(s) (Line 34), a *split* procedure (Line 37) will be invoked. Two sequences of this cluster, which have (relatively) low similarity to each other, are chosen as the seeds of the new clusters.

Finally, if a large number of sequences are labeled as outliers at the end of an iteration, there is a chance that some of these outliers may actually belong to some cluster that fails to be identified. This scenario may happen during the first several iterations of the mining process since the initially picked k clusters may be inadequate to represent all clusters in the database. A new cluster seed will then be picked from outliers. Consequently, the number of clusters increases.

4.6 Complexity Analysis

The entire process terminates when all IPSTs converge. The overall computational time greatly depends on the number of iterations (Line 4 to 43 in Algorithm A.1) actually executed. The computational complexity of each iteration is $O(N \times (k'l^2 + l^2)) = O(N \times k' \times l^2)$ 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 $k' \ll N$.

[2]By definition in Section 2, the average degree of imprecision is equal to the average of the gaps between lower and upper probabilities.

5 Discussion

In this section, we discuss some issues that influence the accuracy and efficiency of CLUSEQ.

5.1 Limited Memory Space

In practice, the memory space is usually limited and the size of each imprecise probabilistic suffix tree is also restricted by the available memory. In CLUSEQ, k trees need to be maintained throughout the entire process, one for each cluster. Each tree can occupy up to $\frac{1}{k}$ of the entire 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 accuracy of similarity estimation.

2. *Prune node with longest path label first.* This is inspired by the *short memory property* that exhibits in many applications. This property implies that the empirical probability distribution on the next symbol given the preceding segments of length L has less chance to differ substantially from the probability distribution conditioning on preceding segments of length greater than L as L increases. Therefore, the pruning of a node with longer path label is expected to have less impact to the similarity estimation (than the pruning of a node with shorter path 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 imprecise probabilistic suffix tree. Let σ and σ' be the path 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.

5.2 Probability Smoothing

The problem of "under sampling" has been observed and a smoothing process was proposed in Bejerano&Yona (1999). The purpose of the smoothing process is to assure that no symbol is predicted to have a zero probability, no matter what suffix is observed before it, even though the empirical count may be zero. This can achieved by enforcing a minimal probability γ_{min} for all symbols and each nonzero empirical probability is decreased such that a total of $n \times \gamma_{min}$ is "collected" to be later shared by all symbols, where n is the number of distinct symbols. The decrement of each empirical probability is done in proportion to its value. Formally, the probability after smoothing can be obtained through the formula $\hat{p} = (1 - n \times \gamma_{min})p + \gamma_{min}$. The proper value of γ_{min} can be determined according to some domain knowledge. We shall mention that this smoothing technique can also be applied to the IPST model to address the under sampling problem.

5.3 Incremental Clustering

If we already have a set of clustered sequences, the computation of clustering new sequences would be efficient. In such a case, we would compare the sequence with the probability suffix tree of each cluster. If the new sequence is similar to a cluster based on the similarity estimation, then this sequence is grouped to that cluster. On the other hand, if no group is similar to the new sequence, the new sequence is classified as an outlier. To take into account of the existence of new clusters, we generate new seed from all outliers after a significant number of new sequences has been classified as outliers. Then we employ the clustering algorithm on the set of outliers to determine whether there exists a new cluster. Due to space limitations, we will omit the detailed discussion of this aspect of CLUSEQ.

6 Results

We implemented the CLUSEQ algorithm in C programming language. All experiments were run on a Sun UltraSparc 10 machine with 128 MB main memory and a 300 MHz CPU.

6.1 Correctness of CLUSEQ Model

The real dataset we applied the CLUSEQ algorithm is a database of 8000 well documented protein randomly sequences selected from the SWISS-PROT data bank. These protein sequences belong to 30 different biological families, single domain and multi-domain families. The size of each family ranges from 140 to 900. The sequence length is between 285 and 895. In this experiment, we want to find the results produced by our algorithm via the label provided by the SWISS-PROT database. The result produced by CLUSEQ with parameters $k = 5$, $c = 10$, $t = 1.25$, $\epsilon = 0.001$ was compared with the standard protein family label and the average length of a significant segment in the probabilistic is 13.8. Table 1 shows the false positives and false negatives of nine families due to space limitations. In this experiment, we treat all these 8000 protein sequences as unlabeled sequences, i.e., no priori knowledge of the number of families, etc..

For all protein families, the corrected labeled proteins are over 90%. It is easy to see that the CLUSEQ algorithm performs well consistently for clusters of diverse sizes. Furthermore, since we initially set $k = 5$ (which is much less that the actual number of clusters), the CLUSEQ algorithm still can successfully identify all clusters with high accuracy, own to the cluster validation procedure. This is very important because the number of clusters is often unknown in advance.

We also compare the results of CLUSEQ with that produced by maximum alignment method (e.g., ProtoMap) for the set of protein sequences in Table 1. We can see that the number of false positives and negatives is very similar between the two approaches. However, CLUSEQ takes much less time (about 420 seonds) than the maximum alignment approach (5600 seonds), which makes CLUSEQ a better choice.

6.2 Effects of ϵ

In our CLUSEQ algorithm, ϵ plays an important role. It does not only effect the results of our algorithm, but also effect the response time of our algorithm, i.e., how fast the results are returned. With larger ϵ, the quality of results degrades, but the response time shortens. Figure 3(a) and (b) shows the average accuracy and response time of the CLUSEQ algorithm with respect to ϵ. Since the average accuracy and average recall over all families are the same, we omit the curve corresponding to the average recall. The test dataset is the same 8000 protein sequences. From the figures, we can see that when the ϵ is sufficiently small, e.g., 10^{-4}, the improvement of the results quality is diminished with smaller ϵ while the response time increases dramatically because more iterations are needed for the convergence of the upper and lower probabilities. Based on this result, if we set $\epsilon = 0.0001$, the results will be sufficiently good and the response time is relatively fast.

6.3 Sensitivity Analysis

To understand the sensitivity of the CLUSEQ algorithm to the sample size m, we experiment with various m on the data set. The convergence threshold is set to $\epsilon = 10^{-4}$. Figure 4 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., average accuracy) 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 the other hand, the response time of the CLUSEQ falls into a valley as shown in Figure 4(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 cluster is very poor and it takes a longer course for CLUSEQ to reach the final clustering.

In addition, we also analyze the sensitivity of CLUSEQ with respect to the significant threshold c. As long as it is within a reasonable range, both the cluster quality and the performance of CLUSEQ show stability to the variance of c. The parameter t controls how similar a sequence should be to its peers in a cluster and its appropriate value is application-dependent and should be specified by an expert. A further study regarding the parameters c and t is left to the full version of the paper.

7 Conclusions

In this paper, we investigated in the problem of automatic clustering of protein sequences. A novel model is proposed for sequence cluster by exploring significant statistical properties possessed by the sequences. The imprecise probabilities are introduced to monitor the convergence of the empirical statistics. This clustering framework can either (1) be applied directly to partition unlabeled sequences into meaning families/groups (without first building models on labeled sequences), or (2) be coupled with some classification tools to refine the classification. An important advantage is that the clustering model can potentially detect unknown (sub)family/(sub)group or even outliers (perhaps due to some unexpected mutations). Biological considerations (such as the a-priori probability distribution of amino acids and the probabilities of amino acids mutation) can also be easily incorporated into the clustering model in a similar manner as discussed in Bejerano&Yona (1999).

References

Apostolico, A. and Bejerano, G. (2000) Optimal amnesic probabilistic automata or how to learn and classify pro-

Table 1. Results for CLUSEQ on Protein Database

Family		ig	pkinase	globin	7tm_1	homeobox	efhand	RuBisCO_large	...	actin	rrm
Size		884	725	681	515	383	320	311	...	142	141
CLUSEQ	False Positive	44	71	14	41	22	30	14	...	4	9
	False Negative	53	8	25	34	27	21	17	...	5	6
Max. Alig.	False Positive	51	60	24	31	17	30	15	...	4	7
	False Negative	43	24	25	30	33	25	13	...	4	7

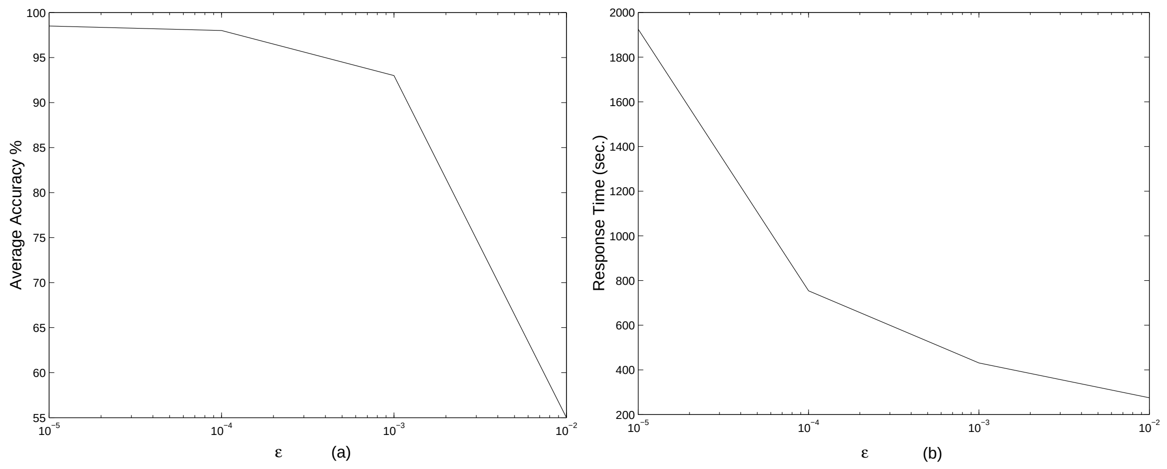


Figure 3. The effects of ϵ

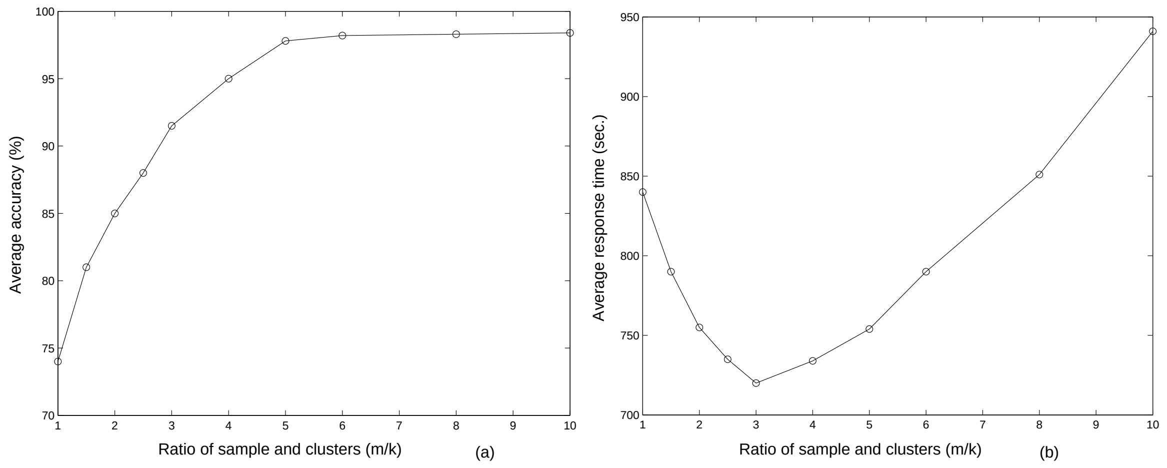


Figure 4. The effects of the number of initial sample

teins in linear time and space. *Proc. of ACM RECOMB*, 25-32.

Bailey, T. and Grundy, W. (1999) Classifying proteins by family using the product of correlated p-values. *Proc. of ACM RECOMB*, 10-14.

The SWISS-PROT protein sequence data bank.

Baldi, P. (2000) On the convergence of a clustering algorithm for protein-coding regions in microbial genomes. *Bioinformatics*, 16(4), 367-371.

Baldi, P. and Baisnee, P. (2000) Sequence analysis by additive scales: DNA structure for sequences and repeats for all lengths. *Bioinformatics*, 16(10), 865-889.

Bejerano, G. and Yona, G. (1999) Modeling protein families using probabilistic suffix trees. *Proc. of ACM RECOMB*, 15-24.

Bejerano, G., Seldin, Y., Margalit, H., and Tishby, N. (2001) Markovian domain fingerprinting: statistical segmentation of protein sequences. *Bioibformatics*, 17(10), 927-934.

Bergeron, A. and Hamel, S. Vector algorithms for approximate string matching. *To appear in Intern. Journal of Foundation of Computer Science*.

Cole, R. and Hariharan, R. (1999) Approximate string matching: a simpler faster algorithm. *Proc. of SODA*, 235-244.

Cole, R. and Hariharan, R. (2000) Faster suffix tree construction with missing suffix links. *Proc. of ACM STOC*, 407-415.

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

Farach, M. (1997) Optimal suffix tree construction with large alphabets. *Proc. of 38th Sym. on FOCS*, 137-143.

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

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

Gusfield, D. (1997) *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press.

Lin, J. (1991) Divergence measures based on the Shannon entropy. *IEEE Tran. on Information Theory*, 37(1), 145-151.

McCreight, E. (1976) A space-economical suffix tree construction algorithm. *J. of ACM*, 23, 262-272.

Ron, D., Singer, Y., Tishby, N. (1996) The power of anmesia: learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3), 117-149.

Silverstein, K., Shoop, E., Johnson, J., and Retzel, E. (2001) MetaFam: a unified classification of protein families. *Bioinformatics*, 17(3), 249-271.

Ukkonen, E. (1995) On-line construction of suffix trees. *Algorithmica*, 14, 249-260.

Walley, P. (1991) *Statistical Reasoning with Imprecise Probabilities*. Chapman and Hall.

Wang, J., Rozen, S., Shapiro, B., Shasha, D., Wang, Z., and Yin, M. New techniques for DNA sequence classification. *J. of Computational Biology*.

Wang, J., Ma, Q., Shasha, D., Wu, C. (2000) Application of neural networks to biological data mining: a case study in protein sequence classification. *Proc. ACM SIGKDD*.

A The CLUSEQ Algorithm

Algorithm A.1 *Algorithm for Sequence Clustering*

```
CLUSEQ($\Sigma$, $k$, $c$, $t$, $\epsilon$)
{ /* $\Sigma$ is a database of sequences;
   $k$ is the minimum number of clusters;
   $c$ is the significance threshold;
   $t$ is the similarity threshold;
   $\epsilon$ is the convergence threshold. */
1:  $T \leftarrow InitialSeed(\Sigma, k, c)$ /* Generate $k$ seeds from $\Sigma$ */
2:  $continue \leftarrow$ **true**
3:  $k' \leftarrow k$ /* $k'$ tracks the actual number of clusters */
4:  **while** $continue$ **do** {
5:      $continue \leftarrow$ **false**
6:      $outliers \leftarrow |\Sigma|$
7:      **for** each sequence $\sigma \in \Sigma$ **do** { /* cluster training */
8:          $\sigma.cluster \leftarrow 0$
            /* $1..k'$ denote cluster IDs while 0 represents outlier */
9:          **for** $j \leftarrow 1$ **to** $k'$ **do**
10:             $sim \leftarrow Similarity(T[j], \sigma, c)$
11:             **if** $sim > t$
12:             **then**
                /* Only the segment in $\sigma$ which delivers similarity
                $sim$ is used in updating $T[j]$ */
13:               $Update(T[j], \sigma)$
14:               $\sigma.cluster \leftarrow j$
15:         **if** $\sigma.cluster > 0$ /* $\sigma$ is similar to at least one cluster */
16:         **then** $outliers \leftarrow outliers - 1$
        }
20:     **for** $j \leftarrow 1$ **to** $k'$ **do** { /*cluster validation */
21:         $gap \leftarrow 0$
22:         $gapnum \leftarrow 0$ /* tracks the number of gaps accumulated */
23:         $convergence \leftarrow$ **true**
24:         **for** each node $x$ in $T[j]$ **do**
25:             **for** each symbol $d$ **do**
26:               $x.\underline{prob}[d] \leftarrow \frac{x.\underline{prob}[d]+x.prob[d]}{2}$
                  /* update lower probability */
27:               $x.\overline{prob}[d] \leftarrow \frac{x.\overline{prob}[d]+x.prob[d]}{2}$
```

```
                    /* update upper probability */
28:                 if $x.count \geq c$
29:                 then /* $x$ is a significant node */
30:                    $gap \leftarrow gap + x.\overline{prob}[d] - x.\underline{prob}[d]$
31:                    $gapnum \leftarrow gapnum + 1$
32:                    if $x.\overline{prob}[d] - x.\underline{prob}[d] > \epsilon$
33:                    then /* $x$ does not converge on symbol $d$ */
34:                       $convergence \leftarrow$ false
35:                       $continue \leftarrow$ true
34:             if $convergence =$ false and $\frac{gap}{gapnum} \geq T[j].gap$
                /* $T[j]$ does not converge yet and the degree
                of imprecision does not improve */
35:             then /* split cluster $T[j]$ */
36:                 $k' \leftarrow k' + 1$
37:                 $Split(\Sigma, T[j], T[k'], c)$
38:             else $T[j].gap \leftarrow \frac{gap}{gapnum}$
            }
39:         if $outliers > |\Sigma| \times 10\%$
40:         then /* A large number of outliers exist */
41:             $k' \leftarrow k' + 1$
42:             $T[k'] \leftarrow AdditionalSeed(\Sigma, c)$
                /* A new cluster is initiated */
43:             $continue \leftarrow$ true
      }
44:   return $T$ /* $T$ contains $k'$ clusters */
}
```