Mining Naturally Smooth Evolution of Clusters from Dynamic Data *

Yi Wang † Shi-Xia Liu‡ Jianhua Feng § Lizhu Zhou¶

Abstract

Many clustering algorithms have been proposed to partition a set of static data points into groups. In this paper, we consider an evolutionary clustering problem where the input data points may move, disappeare, and emerge. Generally, these changes should result in a smooth evolution of the clusters. Mining this naturally smooth evolution is valuable for providing an aggregated view of the numerous individual behaviors.

We solve this novel and generalized form of clustering problem by converting it into a Bayesian learning problem. Analogous to that the EM clustering algorithm clusters static data points by learning a Gaussian mixture model, our method mines the evolution of clusters from dynamic data points by learning a hidden semi-Markov model (HSMM). By utilizing characteristics of the evolutionary clustering problem, we derive a new unsupervised learning algorithm which is much more efficient than the algorithms used to learn traditional variable-duration HSMMs. Because the HSMM models the probabilistic relationship between the dynamic data set and corresponding evolving clusters, we can interpret the learned parameters as the evolving clusters intuitively using the Viterbi filtering technique. Because learning an HSMM is in fact learning an optimal Viterbi filter, the learned cluster evolutions are smooth and fit well with the data. We demonstrate the effectiveness of this method by experiments on both synthetic data and real data.

1 Introduction.

In this paper, we consider a novel and generalized form of clustering problem — clustering a set of dynamic data points, which may move, disappear, and emerge, into evolutionary clusters that also move, disappear, and emerge accordingly. We think this evolutionary clustering problem interesting because many data sets generated by real applications are dynamic, and the evolutionary clusters may provide an aggregated view of the numerous individual behaviors.

A typical example of evolutionary clustering is to detect user communities in a Web forum and to trace the evolutions of these communities. Suppose a Web forum that has d discussion boards and a large number of registered users. Within each day t, there may be a set of, say N_t, active users, denoted by a set $\mathcal{X}_t$, who login and participate discussions. Each of these users, denoted by $\boldsymbol{x}_{t,l} \in \mathcal{X}_t$, can be represented by a data point in a d-dimensional "interest space", where the i-th element of the vector value $\boldsymbol{x}_{t,l}$, denoted by $x_{t,l,i}$, is the normalized frequency that the l-th user posts on the d-th discussion board within day t. By clustering each $\mathcal{X}_t$, we will be able to know the common interests of the active users within day t. By tracing the evolution of the clusters from $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^{T}$, we will have the trends of the common interests.

From this example, we can perceive some general difficulties of the evolutionary clustering problem:

1. Because a user may be active for one day but not for another, the size of $\mathcal{X}_t$, the active users, may vary over t. Generally speaking, the $|\mathcal{X}_t|$ may vary over t.
2. Even if a user is active for several successive days, his/her interest may change during these days, so, generally, the point $\boldsymbol{x}_{t,l}$ may also move over time.
3. Because of (1) and (2), the number of clusters of $\mathcal{X}_t$, denoted by C_t, is also likely to vary over time.
4. Although the interests of some users may change abruptly, their effects to the overall change of common interests are limited, so the clusters usually evolve smoothly.

These difficulties exclude an intuitive and naive solution — partition each $\mathcal{X}_t$ individually into C_t clusters using traditional static clustering methods such as K-means. The exclusion is because that this solution, when clustering $\mathcal{X}_t$, does not consider the temporal coherence between $\mathcal{X}_t$ and its predecessors, $\mathcal{X}_{t-1}$, $\mathcal{X}_{t-2}$, etc. Whereas, clustering static data points is an NP-hard problem and various static clustering algorithms

*This work was finished during Yi Wang's visit at IBM China Research Lab. It was supported in part by the National 973 Program of China under Grant No. 2006CB303103, and the National 863 Program of China under Grant No. 2006AA01A101.

†Department of Computer Science, Tsinghua University and IBM China Research Lab. Email:yi.wang.2005@gmail.com

‡Information Visualization and Interactive Visual Analytics, IBM China Research Lab.

§Department of Computer Science, Tsinghua University.

¶Department of Computer Science, Tsinghua University.

can only find locally optimal clusters. As the naive solution concatenates a series of temporally independent local optima, the generated evolution is often not smooth and cannot reveal the real evolution process.

Another solution that considers the temporal coherence in part makes use of the property that most static clustering algorithms are iterative [9] [6] [20]. This solution uses the clustering result of $\mathcal{X}_{t-1}$ to initialize the iterative clustering of $\mathcal{X}_t$. Because iterative algorithms update the clustering results gradually, the mined cluster evolution may have less abruptions. However, as the clustering of each $\mathcal{X}_t$ converges to a local optimum, whose bias from the global optimum will accumulate with the increasing of t. Although heuristic rules can be used to pull back the tracing from far biased estimation, it is more effective to consider the temporal coherence of the whole sequence $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^T$.

In this paper, we explicitly consider the temporal coherence over $1 \leq t \leq T$ by modeling the underlying stochastic process that generates $\mathbb{X}$ by a hidden semi-Markov model (HSMM) [4]. Analogous to that the EM clustering algorithm [19] [15] converts the problem of clustering a static data, say $\mathcal{X}$, set into learning a Gaussian mixture model from $\mathcal{X}$; in this paper, we show that the clustering of a dynamic data set, $\mathbb{X}$, can be converted into learning an HSMM from $\mathbb{X}$.

We model the output probability density function (pdf) on each hidden state of the HSMM by a Gaussian mixture model, which describes the clusters of an $\mathcal{X}_t \in \mathbb{X}$. So, learning such an HSMM, say with S states, would estimates S clustering configurations[1]. By utilizing characteristics of the evolutionary clustering problem, we derive a new unsupervised learning algorithm which is much more efficient than the algorithms used to learn traditional variable-duration HSMMs.

Then, we can assign each $\mathcal{X}_t \in \mathbb{X}$ to one of these S clustering configurations using the Viterbi filtering technique [4], which garentees that the result sequence of clustering configurations, or equivalently, the evolution of clusters, is the one that *fits best with* the dynamic data set $\mathbb{X}$. If the real evolution is intrinsically smooth, this mined best-fit evolution is also be smooth.

2 Related Work.

As introduced in Section 1, we are addressing an evolutionary clustering problem with a given sequence of data sets, $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^T$. In the area of data mining, there have been three types of sequential-/streaming-data clustering problems under studying. But to the best of our knowledge, the problem addressed by this paper is different from them, and has not been addressed in the literature so far.

The first type of problem under studying is to cluster data points arriving in streaming [1] [2] [3] [14] [8] [10] [12]. This problem covers a wide range of online data analysis application. It differs from our problem in the following aspects: (1) In this problem, only one data point arrives at a time in a stream; whereas in our problem, $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^T$ is a sequence of data sets, and can be considered that a set of N_t data points, $\mathcal{X}_t$, arrives at a time in a sequence. (2) In this problem, the difficulty is the online processing of the streaming data, in which, data points may arrive infinitely and in a fast rate; whereas in our problem, the whole sequence of data sets is given before processing. (3) Although in this problem, the clustering configuration changes over time as in our problem, there is no requirement to consider the temporal coherence of the evolution of clusters.

The second type of problem under studying is to cluster N sequences into C groups [18] [5] [13]. This problem exists primarily in the area of bioinformatics, for example, clustering gene or protein sequences. Similar to our problem, the sequences are given before clustering. But different to our problem, the units to be clustered in this problem are the sequences, but not those data points that constitute the sequences as in our problem. In this problem, only one clustering configuration consisting of C clusters is to be computed; whereas in our problem, there are T clustering configurations to be computed, each for one $\mathcal{X}_t$ in $\mathbb{X}$.

The third type of problem is to cluster N parallel data streams online [6] [9]. This problem is more similar to our problem than the previous two types, because a sequence of clustering configurations are to be computed over time. However, this problem focuses on online data processing, so only those already arrived data sets can be used to compute the current clustering. Whereas, our problem addresses making use of the whole sequence of data sets, $\mathbb{X}$, to direct the clustering of each data set, $\mathcal{X}_t \in \mathbb{X}$, and thus to mine the intrinsically smooth evolution of clusters.

3 Evolutionary Clustering as Learning.

We formulate the evolutionary clustering problem as to input a sequence of static "snapshots" of the dynamic data set, $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^T$, where the $\mathcal{X}_t$'s for every t may have variable sizes, and to output a sequence of clustering configurations, $\mathbb{C} = \{\mathcal{C}_t\}_{t=1}^T$, where each $\mathcal{C}_t$'s may contain variable number of clusters. We solve this problem by learning S clustering configurations from $\mathbb{X}$, and assign each $\mathcal{X}_t$ to one of the S learned clustering configurations.

[1] In this paper, we call the set of clusters of a static data set and related information, such as the weights on each cluster, a "clustering configuration"

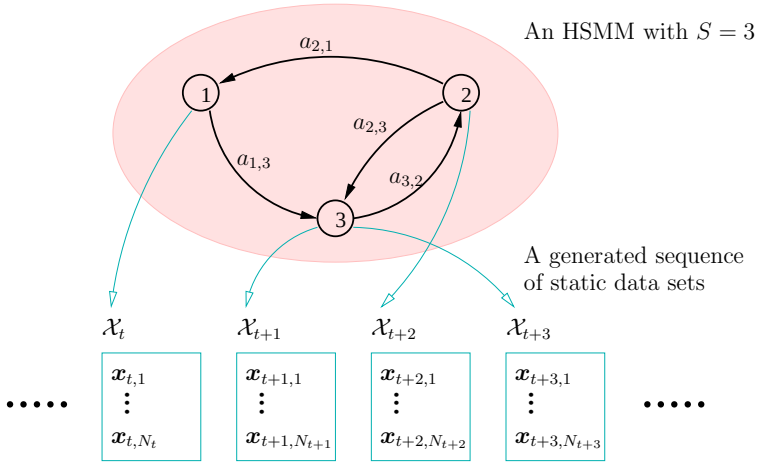

Figure 1: An example HSMM and the sequence of data sets that the HSMM models.

The learning is based on modeling the evolution of clusters by an HSMM with S hidden states, where the output pdf on each state models a clustering configuration. The HSMM models a semi-Markovian process that switches among S hidden states, and whenever it arrives at a new state, a set of *observables* are generated from the output pdf defined on this state. To model the evolutionary clustering problem, we consider that each set $\mathcal{X}_t$ is generated from a state, and thus is corresponded with an output pdf. Therefore, learning the HSMM would give the set of S clustering configurations and the learned probabilistic transitions between the states encode the evolution of clusters. Thus we can use the Viterbi algorithm to align the sequence of $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^T$ to a sequence of output pdfs, which represents the $\mathbb{C} = \{\mathcal{C}_t\}_{t=1}^T$.

In order to make an output pdf models a clustering configuration with C clusters, where within each cluster, the points are relatively closer to each other, we model the j-th output pdfs by a Gaussian mixture model, $p(x)$, with C component distributions,

$$p(x) = \sum_{i=1}^{C} w_{j,c} \cdot p(x \mid \boldsymbol{\lambda}_{j,c}) \ , \tag{3.1}$$

where, $p(x \mid \boldsymbol{\lambda}_{j,c}) = \mathcal{N}(x \mid \mu_{j,c}, \Sigma_{j,c})$ is the c-th Gaussian component, $\boldsymbol{\lambda}_{j,c} = \{\mu_{j,c}, \Sigma_{j,c}\}$ denotes the parameters of the c-th component, including the mean vector, $\mu_{j,c}$, and the covariance matrix, $\Sigma_{j,c}$, and $w_{j,c}$ is the weight to the c-th component that subjects to $\sum_{c=1}^{C} w_{j,c} = 1$.

This technique of modeling static clusters is the key of the EM clustering algorithm. It is based on the observation that data points sampled (or generated) from a Gaussian distribution tend to be around a center point located by the mean vector. Thus a set of data points, $\mathcal{X}$, generated from a Gausssian mixture model with C components, it is likely to form C clusters. The EM clustering algorithm is to learn such a Gaussian mixture model from the input data set $\mathcal{X}$. Because the learning process is to estimate the parameters, $\{w_c, \mu_c, \Sigma_c\}_{c=1}^C$, that makes the model fit $\mathcal{X}$ the best, the result $\{\mu_c\}_{c=1}^C$ just corresponds to the centers of the C clusters.

Similarly, in this paper, as the HSMM with Gaussian mixture output pdfs models the generation process of the dynamic data set, $\mathbb{X}$, we can use the Viterbi filtering technique to interpret the model parameters learned from $\mathbb{X}$ as the evolution of clusters.

The parameters of the HSMM, denoted by $\boldsymbol{\Lambda}$, contains two parts,

$$\boldsymbol{\Lambda} = \{\mathcal{A}, \mathcal{B}\} \ , \tag{3.2}$$

where $\mathcal{A} = \{a_{i,j}\}_{1 \le i,j \le S}$ contains the S^2 transition probabilities,

$$a_{i,j} = P(q_t = j \mid q_{t-1} = i) \ , \tag{3.3}$$

and contains the parameters of the S output pdfs,

$$\mathcal{B} = \{w_{j,c}, \mu_{j,c}, \Sigma_{j,c}\}_{1 \le j \le S, 1 \le c \le C} \ . \tag{3.4}$$

For the clarity of later discussions, we name this HSMM *dc*-HSMM, as it and its learning algorithm are specifically designed for evolutionary clustering of a dynamic data set.

3.1 Why HSMM instead of HMM? Similar to hidden Markov model (HMM), HSMM considers a hidden system that switches over a specified number,

S, of states. In each clock time t, the model switches from a state, say q_{t-1}, to a new state, q_t, according to the transition probability a_{q_{t-1},q_t}.

Different with HMM, after each state switching, an HMM outputs one data point according to an output pdf, whereas HSMM may output more than one data points.

In the evolutionary clustering problem, a set, $\mathcal{X}_t$, with N_t data points is generated after each switching, so HMM is not suitable.

3.2 Model Cluster Configurations with Variable Sizes. As discussed in Section 1, the number of clusters, C_t, may vary over t. However, it is difficult to model various output pdfs with variable number of components accordingly, because an output pdf may be aligned to more than one X_t's (which may contain variable number of clusters) during the Viterbi filtering step.

Thus, mathematically, we assume all output pdfs have the same number, say C, of components. We set $C \approx \max_t\{C_t\}$, and use the weight w_c to qualitatively indicate the credibility of the c-th cluster. When $w_c \to 0$, the c-th cluster is considered not to exist.

4 Learning the Evolution of Clusters.

Given $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^T$, we learn the model parameters $\mathbf{\Lambda}$ by finding a maximum likelihood solution:

$$\mathbf{\Lambda}^* = \operatorname*{argmax}_{\mathbf{\Lambda}} P(\mathbb{X} \mid \mathbf{\Lambda}) \ . \tag{4.5}$$

Without knowing the state from which each $\mathcal{X}_t$ is generated and the component distribution from which each $\boldsymbol{x}_{t,l} \in \mathcal{X}_t$ is generated, it is difficult to write $P(\mathbb{X} \mid \mathbf{\Lambda})$ in an analytical form and to optimize it. So Thus we introduce a variable, q_t $(1 \le q_t \le S)$, for each $\mathcal{X}_t$, to indicate the state that generates $\mathcal{X}_t$, and for each $\boldsymbol{x}_{t,l} \in \mathcal{X}_t$, a variable, $m_{t,l}$ $(1 \le m_{t,l} \le C)$, to indicate the component distribution of the output pdf defined on state q_t that generates $\boldsymbol{x}_{t,l}$.

Using $\boldsymbol{Q} = \{q_t\}_l$ and $\boldsymbol{M} = \{m_{t,l}\}_{t,l}$, where $1 \le l \le N_t$ and $1 \le t \le T$, and carrying out the first-order Markovian dependency, we have

$$\begin{aligned} &P(\mathbb{X}, \boldsymbol{Q}, \boldsymbol{M} \mid \mathbf{\Lambda}) \\ &\quad = \prod_{t=1}^{T} \left[a_{q_{t-1},q_t} \cdot \prod_{l=1}^{N_t} w_{q_t,m_{t,l}} \cdot p(\boldsymbol{x}_{t,l} \mid \boldsymbol{\lambda}_{q_t,m_{t,l}}) \right] \ . \end{aligned} \tag{4.6}$$

Thus we can compute

$$P(\mathbb{X} \mid \mathbf{\Lambda}) = \mathop{\mathbb{E}}_{\boldsymbol{Q},\boldsymbol{M}} P(\mathbb{X}, \boldsymbol{Q}, \boldsymbol{M} \mid \mathbf{\Lambda}) \ . \tag{4.7}$$

Considering $\boldsymbol{Q}$ and $\boldsymbol{M}$ as hidden variables, we can derive an EM algorithm [11] to solve the maximum likelihood problem (4.5). This iterative algorithm alternatively executes an E-step and an M-step, and provably [11] converges to a local optimum.

- **E-step:** An efficient inference process is designed to estimate the probability distribtion of each q_t to the corresponding $\mathcal{X}_t$ and that of each $m_{t,l}$ for the corresponding $\boldsymbol{x}_{t,l}$.

- **M-step:** Model parameters $\mathbf{\Lambda}$ is updated by maximizing an iterative form of the complete-data log-likelihood as required by the proof of convergence of EM algorithm.

4.1 Derivation of the Learning Algorithm. In order to maximize (4.7), the EM algorithm iteratively updates $\mathbf{\Lambda}$ by maximizing an iterative form of the expectation of the complete-date log-likelihood,

$$\begin{aligned} Q(\mathbf{\Lambda} \mid \mathbf{\Lambda}') &= \sum_{\boldsymbol{Q},\boldsymbol{M}} \log P(\mathbb{X}, \boldsymbol{Q}, \boldsymbol{M} \mid \mathbf{\Lambda}) \cdot P(\boldsymbol{Q}, \boldsymbol{M} \mid \mathbb{X}, \mathbf{\Lambda}') \\ &= \sum_{\boldsymbol{Q},\boldsymbol{M}} \left(\sum_{t=1}^{T} \log a_{q_{t-1},q_t} \right) P(\boldsymbol{Q}, \boldsymbol{M} \mid \mathbb{X}, \mathbf{\Lambda}') \ + \\ &\quad \sum_{\boldsymbol{Q},\boldsymbol{M}} \left(\sum_{t=1}^{T} \log b_{q_t}(\mathcal{X}_t) \right) P(\boldsymbol{Q}, \boldsymbol{M} \mid \mathbb{X}, \mathbf{\Lambda}') \end{aligned}$$

where $\mathbf{\Lambda}'$ is the previous estimate of the model parameters.

Carrying out $P(\boldsymbol{Q}, \boldsymbol{M} \mid \mathbb{X}, \mathbf{\Lambda}')$ using the first-order Markovian dependency, we have

$$\begin{aligned} &Q(\mathbf{\Lambda} \mid \mathbf{\Lambda}') = \\ &\sum_{i=1}^{S}\sum_{j=1}^{S}\sum_{t=1}^{T} \log a_{i,j} \cdot P(q_{t-1}{=}i, q_t{=}j \mid \mathbb{X}, \mathbf{\Lambda}') \ + \\ &\sum_{j=1}^{S}\sum_{c=1}^{C}\sum_{t=1}^{T}\sum_{l=1}^{N_t} \log w_{j,c} P(q_t{=}j, m_{t,l}{=}c \mid \mathbb{X}, \mathbf{\Lambda}') \ + \\ &\sum_{j=1}^{S}\sum_{c=1}^{C}\sum_{t=1}^{T}\sum_{l=1}^{N_t} \log p(\boldsymbol{x}_{t,l} \mid \boldsymbol{\lambda}_{j,c}) P(q_t{=}j, m_{t,l}{=}c \mid \mathbb{X}, \mathbf{\Lambda}') \end{aligned} \tag{4.8}$$

Thus the E-step can infer $P(\boldsymbol{Q}, \boldsymbol{M} \mid \mathbb{X}, \mathbf{\Lambda}')$ by estimating

$$\begin{aligned} \xi_t(i,j) &= P(q_{t-1}{=}i, q_t{=}j \mid \mathbb{X}, \mathbf{\Lambda}') \ , \\ \gamma_t(j,c) &= P(q_t{=}j, m_{t,l}{=}c \mid \mathbb{X}, \mathbf{\Lambda}') \ . \end{aligned} \tag{4.9}$$

and the M-step updates $\mathbf{\Lambda}$ by maximizing $Q(\mathbf{\Lambda} \mid \mathbf{\Lambda}')$.

4.2 The E-step. By considering each data set, $\mathcal{X}_t = \{\boldsymbol{x}_{t,l}\}_{l=1}^{N_t}$, as a single observable, the HSMM can be viewed as an HMM, and $\xi_t(i,j)$ can be computed inductively by the Forward-Backward algorithm [16]. In our problem, the *forward-probability* is defined as

$$\alpha_t(j) = P\left(\{\mathcal{X}_k\}_{k=1}^{t},\ \ q_t{=}j \;\middle|\; \boldsymbol{\Lambda}'\right) \ ,$$

and can be computed inductively by:

$$\begin{aligned} \alpha_1(j) &= a(j;\mathbf{start}) \cdot b(\boldsymbol{x}_1; j) \\ \alpha_{t+1}(j) &= \sum_{i=1}^{S} \alpha_t(i) \cdot a_{i,j} \cdot b(\boldsymbol{x}_{t+1}; j) \ , \end{aligned} \tag{4.10}$$

where the constant value, **start**, denotes a virtual state, which is introduced to simplify the model parameters by merging the *initial distribution* into the transition probabilities [21].

The *backward probability* is defined as

$$\beta_t(j) = P\left(\{\mathcal{X}_k\}_{k=t+1}^{T} \;\middle|\; q_t{=}j,\ \boldsymbol{\Lambda}'\right) \ ,$$

and can be computed by:

$$\begin{aligned} \beta_T(j) &= 1 \ , \\ \beta_t(j) &= \sum_{i=1}^{S} a_{i,j} \cdot b(\boldsymbol{x}_{t+1}; i) \cdot \beta_{t+1}(i) \ . \end{aligned} \tag{4.11}$$

These two inductive processes can compute $\alpha_t(j)$'s and $\beta_t(j)$'s for all $1 \le t \le T$ and $1 \le j \le S$, so $\xi_t(i,j)$ can be computed by,

$$\begin{aligned} \xi_t(i,j) &= \frac{P(q_{t-1}{=}i, q_t{=}j, \mathbb{X}, \boldsymbol{\Lambda}')}{P(\mathbb{X} \mid \boldsymbol{\Lambda}')} \\ &= \frac{\alpha_{t-1}(i) \cdot a_{i,j} \cdot b(\mathcal{X}_t; j) \beta_t(j)}{\sum_{1 \le j \le S} \alpha_T(j)} \ . \end{aligned} \tag{4.12}$$

Computing $\gamma_t(j,c)$ can also use the $\alpha_t(j)$'s and $\beta_t(j)$'s, because

$$\gamma_t(j,c) = \gamma_t(c \mid j) \cdot \gamma_t(j) \ , \tag{4.13}$$

where,

$$\begin{aligned} \gamma_t(j) &= P(q_t{=}j \mid \mathbb{X}, \boldsymbol{\Lambda}') \\ \gamma_t(c \mid j) &= P(m_{t,l}{=}c \mid q_t{=}j, \mathbb{X}, \boldsymbol{\Lambda}') \ , \end{aligned}$$

in which,

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(\mathbb{X} \mid \boldsymbol{\Lambda}')} = \frac{\alpha_t(j)\beta_t(j)}{\sum_{1 \le j \le S} \alpha_T(j)} \ . \tag{4.14}$$

The $\gamma_t(c \mid j)$ can be computed using the Bayes' rule if we consider the weight $w_{j,c}$ in (3.1) as the prior probability of the c-th mixture component:

$$\gamma_t(c \mid j) = \frac{w'_{j,c} \cdot p(\boldsymbol{x}_{t,l} \mid \boldsymbol{\lambda'}_{j,c})}{\sum_{c=1}^{C} w'_{j,c} \cdot p(\boldsymbol{x}_{t,l} \mid \boldsymbol{\lambda'}_{j,c})} \ . \tag{4.15}$$

4.3 The M-step. To derive the updating rule for $a_{i,j}$, we solve the equation $\partial Q / \partial a_{i,j} = 0$. Using (4.8) and introducing the Lagrange multiplier η with the constraint that $\sum_j a_{i,j} = 1$, we have

$$\frac{\partial}{\partial a_{i,j}} \left[\sum_{t=1}^{T} \log a_{i,j} \cdot \xi_t(i,j) + \eta \left(\sum_{j=1}^{S} a_{i,j} - 1 \right) \right] = 0$$

or

$$\frac{1}{a_{i,j}} \sum_{t=1}^{T} \xi_t(i,j) + \eta = 0 \ .$$

Summing both sides over $1 \le j \le S$, we get $\eta = -\sum_{t=1}^{T} \xi_t(i,j) = -\gamma_t(i)$ and

$$a_{i,j} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)} \ . \tag{4.16}$$

To update $w_{j,c}$, we solve the equation $\partial Q / \partial w_{j,c} = 0$. Using (4.8) and introducing the Lagrange multiplier ζ with the constraint that $\sum_{c=1}^{C} w_{j,c} = 1$, we have

$$\frac{\partial}{\partial w_{j,c}} \left[\sum_{t=1}^{T} \sum_{l=1}^{N_t} \log w_{j,c} \cdot \gamma_t(j,c) + \zeta \left(\sum_{c=1}^{C} w_{j,c} - 1 \right) \right] = 0 \ .$$

Solving this equation results in $\zeta = -\sum_l \sum_t \gamma_t(j)$ and

$$w_{j,c} = \frac{\sum_{t=1}^{T} \sum_{l=1}^{N_t} \gamma_t(j,c)}{\sum_{t=1}^{T} \sum_{l=1}^{N_t} \gamma_t(j)} \ . \tag{4.17}$$

To update $\boldsymbol{\lambda}_{j,c} = \{\mu_{j,c}, \Sigma_{j,c}\}$, we solve the equation $\partial Q / \partial \boldsymbol{\lambda}_{j,c} = 0$. Taking the derivative of (4.8) with respect to $\mu_{j,c}$ and $\Sigma_{j,c}$ respectively, setting to zero, and solving the equations, we get,

$$\begin{aligned} \mu_{j,c} &= \frac{\sum_{t=1}^{T} \sum_{l=1}^{N_t} \boldsymbol{x}_{t,l} \cdot \gamma_t(j,c)}{\sum_{t=1}^{T} \sum_{l=1}^{N_t} \gamma_t(j,c)} \ , \\ \Sigma_{j,c} &= \frac{\sum_{t=1}^{T} \sum_{l=1}^{N_t} \gamma_t(j,c) \cdot (\boldsymbol{x}_{t,l} - \mu_{j,c})(\boldsymbol{x}_{t,l} - \mu_{j,c})^T}{\sum_{t=1}^{T} \sum_{l=1}^{N_t} \gamma_t(j,c)} \ . \end{aligned} \tag{4.18}$$

5 Interpreting the Evolution of Clusters.

Given the learned knowledge, particularly, the estimated $\boldsymbol{\Lambda}^*$, we can interpret the sequence of data sets, $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^{T}$, as a sequence of clustering configurations, $\mathbb{C} = \{\mathcal{C}_t\}_{t=1}^{T}$, where $\mathcal{C}_t$ changes smoothly over t.

As discussed in Section 3, each output pdf of the learned HSMM represents a clustering configuration, and learning the model with S states had estimated S clustering configurations and their transition probabilities (3.3). So, given $\boldsymbol{\Lambda}^*$ and $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^{T}$, we are to

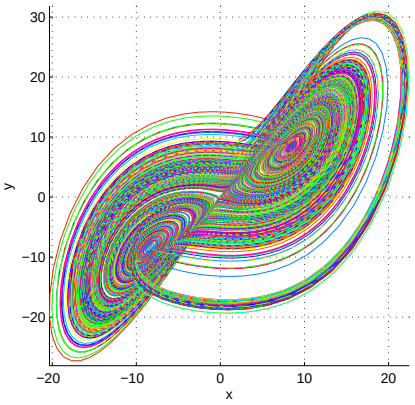

Figure 2: The trajectory of a set of data points that move chaotically.

align each $\mathcal{X}_t \in \mathbb{X}$ to a state $\hat{q}_t$, which corresponds to a clustering configuration that fits $\mathcal{X}_t$ optimally.

A straight-forward solution may be finding the state $\hat{q}_t$ for each $\mathcal{X}_t$ where,

$$\hat{q}_t = \underset{1\le i\le S}{\operatorname{argmax}} \sum_{l=1}^{N_t} \log p(\boldsymbol{x}_{t,l} \mid \boldsymbol{\lambda}_i) \quad . \tag{5.19}$$

However, this solution does not consider the temproal coherence over the $\hat{\boldsymbol{Q}} = \{\hat{q}_t\}_{t=1}^T$.

Alternatively, in our problem, it is possible to consider the learned HSMM, $\boldsymbol{\Lambda}^*$, as an HMM by taking each $\mathcal{X}_t$ as a single observable. Then the Viterbi algorithm can be used to align the training sequence, $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^T$, to a path of HSMM states $\hat{\boldsymbol{Q}} = \{\hat{q}_t\}_{t=1}^T$. Because the Viterbi algorithm actually maximizes the likelihood between the state path $\hat{\boldsymbol{Q}}$ and $\mathbb{X}$,

$$\hat{\boldsymbol{Q}} = \underset{\boldsymbol{Q}}{\operatorname{argmax}}\, P(\hat{\boldsymbol{Q}} \mid \mathbb{X}, \boldsymbol{\Lambda}^*) \quad , \tag{5.20}$$

so $\hat{\boldsymbol{Q}}$ is the one, along which, $\mathbb{X}$ is the most likely been generated. In other words, $\hat{\boldsymbol{Q}}$ encodes the clustering evolution process that fits best with $\mathbb{X}$.

For each state $\hat{q}_t$, the associated output pdf is defined by the parameter set, $\boldsymbol{\lambda}_{\hat{q}_t}$, which includes the parameters that define C components:

$$\boldsymbol{\lambda}_{\hat{q}_t} = \{w_{\hat{q}_t,c}, \mu_{\hat{q}_t,c}, \Sigma_{\hat{q}_t,c}\}_{c=1}^C \quad . \tag{5.21}$$

According to the definition of the HSMM in Section 3, $\mu_{j,c}$ is the center of the c-th cluster, $\Sigma_{j,c}$ determines the shape of the c-th cluster, and $w_{j,c}$ denotes the importance of the c-th cluster, measured by the expected number of training data points that are generated by the cluster.

For two reasons, this mined evolutionary clustering is the one that fits the evolution of $\mathbb{X}$ best, and takes the complete temproal coherence over the whole time axis, $1 \le t \le T$, under consideration:

1. The HSMM, $\boldsymbol{\Lambda}^*$, is learned by maximizing the likelihood and makes use of the whole sequence of $\mathbb{X}$ to estimate the complete temporal coherence; and
2. the Viterbi alignment is provably [17] an optimal filtering process which also maximizes the likelihood and takes the complete temporal coherence under consideration.

6 Discussions.

6.1 Tuning Parameters. Similar to most hidden Markovian models, including kinds of HMMs and HSMMs, most parameters of the *dc*-HSMM are learned automatically, except that the number of hidden states, S, and the number of component distributions in the output pdfs, C, have to be specified before learning. Fortunately, it is not difficult to determine the proper values of S and C.

In our problem, S has an explicit meaning of the number of learned clustering configurations, each of which may be corresponded to one or more data sets, $\mathcal{X}_t$, in the Viterbi alignment step. So, the larger the S, the less the data sets are corresponded to a same clustering configuration, and the more precise and the more smooth the mined evolution is. Ideally, when $S \approx T$, almost every $\mathcal{X}_t$ will be corresponded to one learned clustering configuration, so the mined evolution should be very smooth.

Although, for most HMMs and HSMMs, large S as $S \approx T$ would increase the number of output pdfs and transition probabilities so much that it often makes the given amount of training data not enough to support a sufficient statistics, or unbiased learning, of the model. Fortunately, for our problem, because each element in the training sequence is not a single data point, but a set of data points, $\mathcal{X}_t$, which contains N_t data points, $\mathcal{X} = \{\boldsymbol{x}_{t,l}\}_{l=1}^{N_t}$, and N_t is usually a large number that makes the clustering problem meaningful. Thus, setting $S \approx T$ won't cause insufficient statistics. In practice, we can even set S to be larger than T to learn a very smooth evolution.

With S fixed, there leaves only one parameter, C, to be determined. As C is a scalar parameter, many commonly used methods can be used to automatically determine C by maximizing the Bayesian Information Criterion (BIC) [7] with respect to C.

6.2 Algorithmic Complexity. The complexity of the learning algorithm involves those of the E-step and

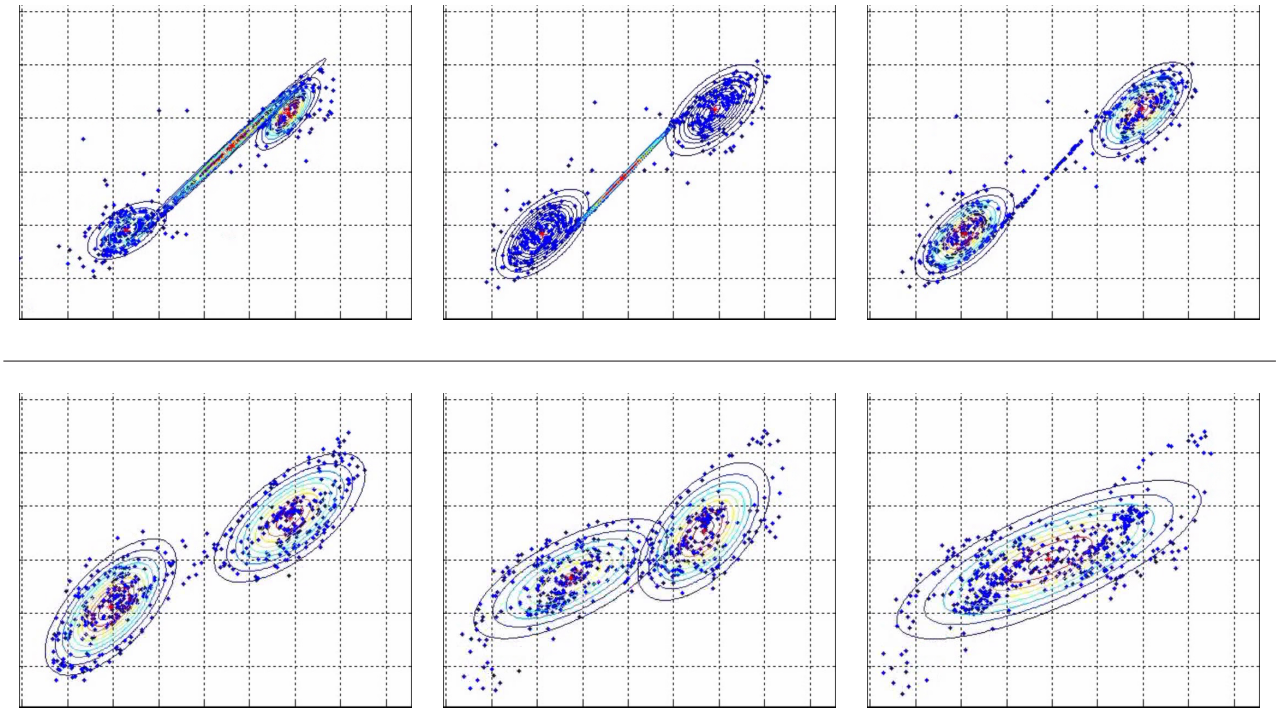

Figure 3: The evolution of clusterings mined from a set of chaotically dynamic data points.

the M-step. The E-step invokes the Forward-Backward algorithm (4.10) (4.11) to compute $\alpha_t(j)$ and $\beta_t(j)$. This step consumes time complexity of $O(S^2T)$.

In addition, computing $\xi_t(i,j)$ takes $O(S^2T)$ and computing $\gamma_t(j,c)$ takes $O(SCT)$. Because C, the number of mixture components in the output pdf, is usually much smaller than S, the number of hidden states, the overall complexity of the E-step is $O(S^2T)$.

As the M-step computes the $S \times S$ transition probabilities according to(4.16) and the $S \times C$ output pdfs according to (4.17) and (4.18), it takes $O(S^2)$. Combining $O(S^2T)$ and $O(S^2)$, the overall complexity of the learning algorithm is $O(S^2T)$, which is much smaller than the complexity of traditional HSMM learning algorithm, $O(S^2C^2T)$. A comparison is in following discussion.

The complexity of the Viterbi filtering algorithm is $O(S^2T)$, which is independent with C because, as discussed in Section 5, the Viterbi filtering is carried out by considering each data set $\mathcal{X}_t$ a single observable.

6.3 Comparisons with Other Models. An HMM with Gaussian mixture output pdfs [16] could be learned if we sequentialize each data set $\mathcal{X}_t$ so that $\mathbb{X}$ is turned into a sequence of data points, $\{\boldsymbol{x}_{t,l}\}_{t,l}$, from a sequence of data sets, $\{\mathcal{X}_t\}_t$. This would also generate a set of Gaussian mixtures. However, these Gaussian mixtures do not represent clustering configurations of the $\mathcal{X}_t$'s, because during learning the HMM, two independent hidden variables, $q_{t,l}$ and $m_{t,l}$ are introduced for each $\boldsymbol{x}_{t,l}$ to indicate the state and component that generate $\boldsymbol{x}_{t,l}$. This breaks the constraint that all $\boldsymbol{x}_{t,l} \in \mathcal{X}_t$ are generated by the same mixture model.

The *dc*-HSMM model also differs from most existed HSMM models, which usually fall into two categories: the fixed duration HSMM and the variable duration HSMM, where *duration* means the number of successive data points that are generated from a hidden state at a time. For the fixed duration HSMM, the duration of each state is fixed. For example, the output pdf on state i always generates a fixed-size set of N_i data points in each time. But for our problem, a state should be able to generate different sets of $\mathcal{X}_t$ with variable sizes, so the durations of *dc*-HSMM states are not fixed.

However, the *dc*-HSMM also differs from the variable duration HSMM. Learning of a variable duration HSMM needs to estimate all possible durations or the expected duration of each state, whereas learning *dc*-HSMM does not need to, because the clustering problem has an interesting property that we know explicitly the size of each data set $\mathcal{X}_t$ is N_t, thus we know that every N_t successive data points are generated by the same state. Our algorithm uses this property by introducing the hidden variable q_t for each data set $\mathcal{X}_t$, and then

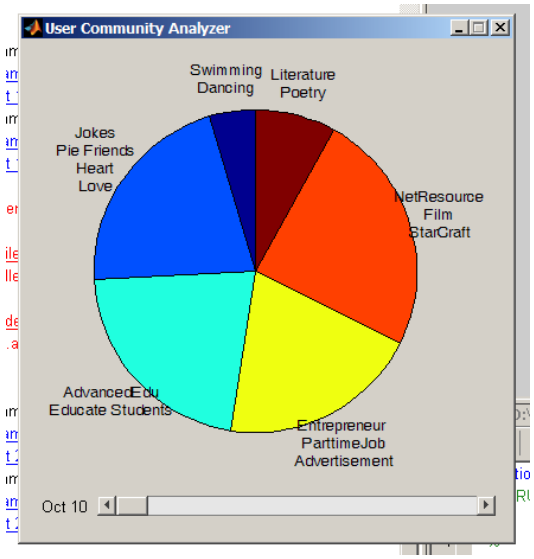

Figure 4: An interactive viewer of the mined evolution of Web communities.

introducing an additional $m_{t,l}$ for each q_t. So, different with the E-step of traditional variable duration HSMM learning algorithms, which infers two independent hidden variables, $q_{t,l}$ and $m_{t,l}$, and reaches the complexity of $O(S^2C^2T)$, our algorithm infers q_t by computing $\gamma_t(q_t)$, and then for each q_t, infers $m_{t,l}$ by computing $\gamma_t(q_t, m_{t,l})$, which leads to a much less complexity of $O(S^2T)$.

7 Experiments.

We first test the *dc*-HSMM approach on synthetic data to mine the evolution of clusters formed by a set of chaotically moving data points, and then, we demonstrate the *dc*-HSMM by a real applications of mining the evolution of user communities from a Web forum.

Both of the experiments use an implementation of the *dc*-HSMM learning algorithm in C++. In order to ensure the precision for long training sequences, our implementation computes $\xi_t(i,j)$ and $\gamma_t(j,c)$, (4.9), in a very high precision level of 50 using the MAPL library (`http://www.tc.umn.edu/~ringx004/mapm-main.html`).

7.1 Dynamically Clustering Chaotic Processes.

In this experiment, we consider a dynamic set of data points, where each data point moves along a Lorenz chaotic process in 3D space:

$$\begin{aligned} \mathrm{d}x/\mathrm{d}t &= a(y-x) \\ \mathrm{d}y/\mathrm{d}t &= x(b-z)-y \\ \mathrm{d}z/\mathrm{d}t &= xy-cz \end{aligned} \tag{7.22}$$

We generate the testing data set by updating the positions of these data points every $\mathrm{d}t = 0.01$ for 1000 steps.

Initially, the testing dynamic data set includes 500 points, whose positions are initialized at random. During the process, some data points die and disappear whereas some others are born at random initial positions. The death and born of data points are at random.

The positions of all living data points at time t form the set $\mathcal{X}_t$. The $T = 1000$ steps simulation results in an $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^{T=1000}$. Denoting the size of $\mathcal{X}_t$ by N_t, during the 1000 steps, the minimum N_t is 329, the maximum N_t is 578, and the average value, $\sum_{t=1}^{T=1000} N_t/T$, is 497.

Figure 2 shows the trajectories of these data points with the factors in (7.22) set as $a = 10$, $b = 28$ and $c = 8/3$. We also rendered an animation of the movement of the dynamic data points. It can be seen from the animation that as the data points move gradually, when they are alive, they form clusters that change smoothly.

Learning the sequence of 1000 data sets with parameters $S = 500$ and $C = 10$ on a Pentium IV 2.4GHz computer takes about 5 minutes. Using the mining algorithm discussed in Section 5, we mined the smooth evolution of the clusters and rendered the result together with the moving points into an animation. We select a segment of the animation for fast download: `http://dbgroup.cs.tsinghua.edu.cn/wangyi/dynacluster/example_1.mpg`. This segment contains some typical evolution behaviors, including the vanishing, merging, and separation of existed clusters, as well as the emergence of new clusters.

Figure 3 shows some frames selected from the animation. The top row of three frames, from left to right, shows a process that the data points are separated into two clusters. In the left frame, the data points are relatively closer and form three overlapping clusters; in the middle frame, one of these clusters becomes smaller; and in the right frame, it vanishes. The bottom row of three frames shows a merging process of two separated clusters.

7.2 Evolution of User Communities.

To test our method on real applications, we mine the evolution of user communities of a Web forum, `http://bbs.smth.org`. This forum has over 200 discussion boards, each labeled by a title indicating the topic of this board. In the past five years, this Web forum attracts over 33,000

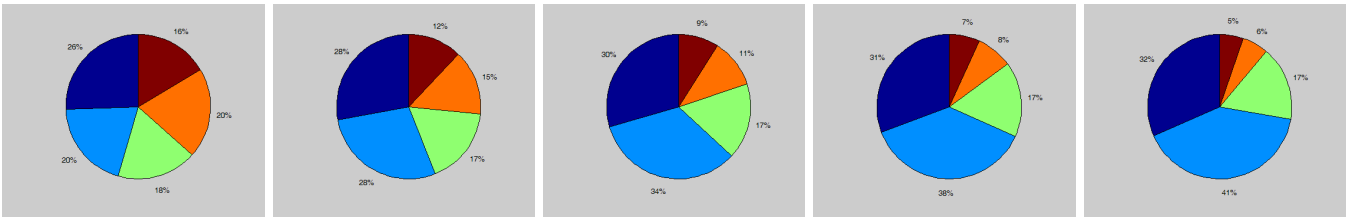

Figure 5: A sequence of frames showing the smooth change of sizes of several Web user communities.

of its totally 200,000 registered users everyday, mainly university students, to login and participate discussions. According to the number of new posts posted in the past one year onto each of the over 200 discussion boards, we selected 16 hottest boards and measure the *interests* of a user by the numbers of his/her posts on these 16 boards.

Due to the huge number of users, it becomes valuable to discover the user communities of the Web forum and their evolution over time. For each day t, we select a set of about 2,000 users, $\mathcal{X}_t$, who had posted the most on the selected 16 boards. Each element, $\boldsymbol{x}_{t,l} \in \mathcal{X}_t$, is the interests of a user and is represented by a 16-dimensional vector, where the value of the d-th dimension, $x_{t,l,d}$, is the number of posts of this user on the d-th selected board during day t.

We traced the posts that were added during the past 360 days by scanning the background relational database. The authors of these posts, represented by their interests vectors, form a sequence of data sets, $\mathbb{X} = \{\mathcal{X}_t\}_{t=1}^{T=360}$. Using the learning and mining method discussed in this paper, we mined the naturally smooth evolution of user interests clusters, or communities, during these 360 days.

However, the mining result can not be visualized as the previous experiment, because the dimensionality, 16, is too large to be represented comprehensively in 2D or 3D. We had tried to project the data points and the mined clusters into lower-dimensional space, but too often the resulted layout of clusters does not utilize the screen space economically.

Therefore, alternative to visualizing the clusters in the 16D interests space or its low-dimensional projections, we visualize only the relative sizes of the clusters by a comprehensive dynamic pie chart, whose each sector corresponds to a cluster and with size changes over time. Given any time t, the area of the c-th sector is proportional to the importance of the c-th cluster of the clustering configuration of $\mathcal{X}_t$: the $w_{q_t,c}$ in (3.1). Using the visualization method of dynamic pie chart, we developed an interactive visualization application in MATLAB allowing the users to explore the evolution of the user communities over time, as well as to view the communities of any specified date.

Figure 4 shows a snapshot of the program, which visualizes the user communities in Oct. 10, 2005. This program labels each sector by the names of the discussion boards that are accessed most frequently by the users of the corresponding community. Figure 5 shows several successive frames animated by this program, which presents the smooth evolution of the user communities during several successive days.

8 Conclusions.

In this paper, we propose to solve a novel and interesting clustering problem: mining the smooth evolution of clusters that are formed by a dynamic data set. As discussed in Section 2, this problem is different from the dynamic clustering problems under studying.

We solve the problem by converting it into a Bayesian learning problem. We design a hidden semi-Markov model, the *dc*-HSMM, to model the underlying stochastic process that generates the dynamic set of data points. Representing the dynamic data set by a sequence of static snapshots, we develop an efficient learning algorithm to estimate the model. Given the learned model, the naturally smooth evolution of the clusters can be mined by the Viterbi filtering technique.

We demonstrate the value and flexibility of this method by two experiments. One mines a dynamic data set synthesized from chaotic processes, the other mines the evolution of user communities of a real Web forum. We also propose accompanying visualization methods to present the mined smooth evolution intuitively and comprehensively.

References

[1] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip Yu. A framework for clustering evolving data streams. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 2003.

[2] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip Yu. A framework for projected clustering of high dimensional data streams. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 2004.

[3] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip Yu. On demand classification of data streams. In *Proc. ACM Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, 2004.

[4] Mehran Azimi, Panos Nasiopoulos, and Rabab Kreidieh Ward. Offline and online identification of hidden semi-markov models. *IEEE Trans. On Signal Processing*, 53(8):2658–2663, 2005.

[5] Ziv Bar-Joseph, Georg Gerber, David K. Gifford, Tommi S. Jaakkola, and Itamar Simon. A new approach to analyzing gene expression time series data. In *Proc. Intel. Conf. on Computational Biology*, pages 39–48, 2002.

[6] Jürgen Beringer and Eyke Hüllermeier. Online clustering of parallel data streams. *Data and Knowledge Engineering*, 2005.

[7] Christopher M. Bishop. *Neural networks for pattern recognition.* Oxford University Press, Oxford, UK, 1996.

[8] Feng Cao, Martin Estery, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *Proc. SIAM Conf. on Data Mining (SDM)*, 2006.

[9] Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. Evolutionary clustering. In *Proc. ACM Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 554–560, 2006.

[10] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, 2003.

[11] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J. Royal Statistical Society Series B*, 39:1–38, 1977.

[12] Pedro Domingos and Geoff Hulten. A general method for scaling up machine learning algorithms and its application to clustering,. In *Proc. Intl. Conf. on Machine Learning (ICML)*, 2001.

[13] C. S. Möller-Levet, F. Klawonn, K.-H. Cho, H. Yin, and O. Wolkenhauer. Clustering of unevenly sampled gene expression time-series data. *Fuzzy Sets and Systems*, 152(1):49–66, 2005.

[14] Liadan O'Callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. Streaming-data algorithms for high-quality clustering. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, 2002.

[15] Carlos Ordonez and Edward Omiecinski. Frem: fast and robust em clustering for large data sets. In *Proc. ACM Conf. on Information and Knowledge Management (CIKM)*, 2002.

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

[17] Nicholas D. Sidiropoulos. The Viterbi optimal runlength-constrained approximation nonlinear filter. *IEEE Trans. On Signal Processing*, 44(3):586–598, 1996.

[18] Jarke J. van Wijk and Edward R. van Selow. Cluster and calendar based visualization of time series data. In *IEEE Sym. on Information Visualization (InfoVis)*, 1999.

[19] Jakob J. Verbeek, Jan R. Nunnink, and Nikos Vlassis. Accelerated em-based clustering of large data sets. *Data Mining and Knowledge Discovery*, 13(3):291–307, 2006.

[20] Yi Wang and et al. Detecting and tracking the evolution of user communities from dynamic social network. In *submitted to WWW'07*, 2006.

[21] Andrew D. Wilson and Aaron F. Bobick. Parametric hidden Markov models for gesture recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(9):884–900, 1999.