

Density-Based Clustering over an Evolving Data Stream with Noise

Feng Cao ^{*} Martin Ester[†] Weining Qian [‡] Aoying Zhou [§]

Abstract

Clustering is an important task in mining evolving data streams. Beside the limited memory and one-pass constraints, the nature of evolving data streams implies the following requirements for stream clustering: no assumption on the number of clusters, discovery of clusters with arbitrary shape and ability to handle outliers. While a lot of clustering algorithms for data streams have been proposed, they offer no solution to the combination of these requirements. In this paper, we present *DenStream*, a new approach for discovering clusters in an evolving data stream. The “dense” micro-cluster (named core-micro-cluster) is introduced to summarize the clusters with arbitrary shape, while the potential core-micro-cluster and outlier micro-cluster structures are proposed to maintain and distinguish the potential clusters and outliers. A novel pruning strategy is designed based on these concepts, which guarantees the precision of the weights of the micro-clusters with limited memory. Our performance study over a number of real and synthetic data sets demonstrates the effectiveness and efficiency of our method.

Keywords: Data mining algorithms, Density based clustering, Evolving data streams.

1 Introduction

In recent years, a large amount of streaming data, such as network flows, sensor data and web click streams have been generated. Analyzing and mining such kinds of data have been becoming a hot topic [1, 2, 4, 6, 10, 14]. Discovery of the patterns hidden in streaming data imposes a great challenge for cluster analysis.

The goal of clustering is to group the streaming data into meaningful classes. The data stream for mining often exists over months or years, and the underlying model often changes (known as evolution) during this time [1, 18]. For example, in network monitoring, the

TCP connection records of LAN (or WAN) network traffic form a data stream. The patterns of network user connections often change gradually over time. In environment observation, sensors are used to monitor the pressure, temperature and humidity of rain forests, and the monitoring data forms a data stream. A forest fire started by lightning often changes the distribution of environment data.

Evolving data streams lead to the following requirements for stream clustering:

1. No assumption on the number of clusters. The number of clusters is often unknown in advance. Furthermore, in an evolving data stream, the number of natural clusters is often changing.
2. Discovery of clusters with arbitrary shape. This is very important for many data stream applications. For example, in network monitoring, the distribution of connections is usually irregular. In environment observation, the layout of an area with similar environment conditions could be any shape.
3. Ability to handle outliers. In the data stream scenario, due to the influence of various factors, such as electromagnetic interference, temporary failure of sensors, weak battery of sensors, etc., some random noise appears occasionally.

We note that since data stream applications naturally impose a limited memory constraint, it becomes more difficult to provide arbitrary-shaped clustering results using conventional algorithms. (1) There is no global information about data streams, which is often required in conventional density based algorithms [3, 8, 13]. (2) Clusters with arbitrary shape are often represented by all the points in the clusters [8, 12, 13], which is often unrealistic in stream applications due to the memory constraint. (3) Previously proposed streaming algorithms, e.g., CluStream [1], often produce spherical clusters, because the distance is adopted as the measurement.

Furthermore, because of the dynamic nature of evolving data streams, the role of outliers and clusters are often exchanged, and consequently new clusters often emerge, while old clusters fade out. It becomes

^{*}caofeng@fudan.edu.cn, Department of Computer Science and Engineering, Fudan University.

[†]ester@cs.sfu.ca, School of Computing Science, Simon Fraser University.

[‡]wnqian@fudan.edu.cn, Department of Computer Science and Engineering, Fudan University.

[§]ayzhou@fudan.edu.cn, Department of Computer Science and Engineering, Fudan University.

more complex when noise exists. For example, in CluStream [1], the algorithm continuously maintains a fixed number of micro-clusters. Such an approach is especially risky when the data stream contains noise. Because a lot of new micro-clusters will be created for the outliers, many existing micro-clusters will be deleted or merged. Ideally, the streaming algorithm should provide some mechanism to distinguish the seeds of new clusters from the outliers.

In this paper, we propose *DenStream*, a novel algorithm for discovering clusters of arbitrary shape in an evolving data stream, whose salient features are described below.

- The core-micro-cluster synopsis is designed to summarize the clusters with arbitrary shape in data streams.
- It includes a novel pruning strategy, which provides opportunity for the growth of new clusters while promptly getting rid of the outliers. Thus, the memory is limited with the guarantee of the precision of micro-clusters.
- An *outlier-buffer* is introduced to separate the processing of the potential core-micro-clusters and outlier-micro-clusters, which improves the efficiency of DenStream.
- Due to its adaptability to the change of clusters and the ability to handle outliers in data streams, DenStream achieves consistently high clustering quality.

The remainder of this paper is organized as follows: Section 2 surveys related work. Section 3 introduces basic concepts. In Section 4, we propose the DenStream algorithm. In Section 5, we discuss some detailed techniques of DenStream. Section 6 describes the performance study. Section 7 concludes this paper.

2 Related work

Various methods for discovering clusters of arbitrary shape have been proposed in the literature [3, 8, 12, 13, 17]. These methods assume that all the data are resident on hard disk, and one can get global information about the data at any time. Hence, they are not applicable for processing data streams which require a one-pass scan of data sets and clustering via local information. IncrementalDBSCAN [9] is an incremental method for data warehouse applications; however, it can only handle a relatively stable environment but not fast changing streams. In particular, it cannot deal with the case with limited memory.

Recently, the clustering of data streams has been attracting a lot of research attention. Previous methods, one-pass [4, 10, 11] or evolving [1, 2, 5, 18], do not consider that the clusters in data streams could be of arbitrary shape. In particular, their results are often spherical clusters.

One-pass methods typically make the assumption of the unique underlying model of the data stream, i.e., they cannot handle evolving data distributions. These algorithms [4, 10, 11] adopt a divide-and-conquer strategy and achieve a constant-factor approximation result with small memory. A subroutine called LO-CALSEARCH is performed every time when a new chunk arrives to generate the cluster centers of the chunk. The VFKM algorithm [7] extends the k -means algorithm by bounding the learner's loss as a function of the number of examples in each step.

The evolving approaches view the behavior of the stream as it may evolve over time. The problem with CluStream [1] is the predefined constant number of micro-clusters. For example, suppose it is set to ten times of the number of input clusters [1], different "natural" clusters may be merged. Moreover, since a variant of k -means is adopted to get the final result, a "natural" cluster may be split into two parts, because the distance is adopted as the measurement. HPStream [2] introduces the concept of projected cluster to data streams. However, it cannot be used to discover clusters of arbitrary orientations in data streams. Although some simple mechanism is introduced in [1, 2] to deal with outliers, the outliers still greatly influence their formation of micro-clusters. An artificial immune system based clustering approach was proposed in [15], but they didn't give the memory bound which is important for stream applications. The problem of clustering on multiple data streams was addressed in [5, 18].

3 Fundamental Concepts

Cluster partitions on evolving data streams are often computed based on certain time intervals (or windows). There are three well-known window models: landmark window, sliding window and damped window.

We consider the problem of clustering a data stream in the damped window model, in which the weight of each data point decreases exponentially with time t via a fading function $f(t) = 2^{-\lambda t}$, where $\lambda > 0$. The exponentially fading function is widely used in temporal applications where it is desirable to gradually discount the history of past behavior. The higher the value of λ , the lower importance of the historical data compared to more recent data. And the overall weight of the data stream is a constant $W = v(\sum_{t=0}^{t=t_c} 2^{-\lambda t}) = \frac{v}{1-2^{-\lambda}}$, where t_c ($t_c \rightarrow \infty$) is the current time, and v denotes the speed

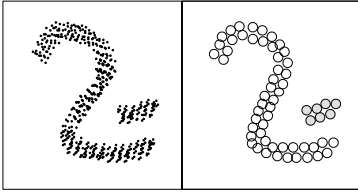


Figure 1: Representation by c-micro-clusters

of stream, i.e., the number of points arrived in one unit time.

In static environment, the clusters with arbitrary shape are represented by all the points which belong to the clusters. Therefore, a naive approach would be to maintain all the points in memory. When a clustering request arrives, these points could be clustered by the DBSCAN algorithm [8] to get the final result. The clustering result is a group of (weighted) core objects C (see Definition 3.1) with corresponding cluster labels, which guarantee that the union of the ϵ neighborhood of C cover the density area (see Definition 3.2).

DEFINITION 3.1. (*core object*) A core object is defined as an object, in whose ϵ neighborhood the overall weight of data points is at least an integer μ .

DEFINITION 3.2. (*density-area*) A density area is defined as the union of the ϵ neighborhoods of core objects.

However, it is unrealistic to provide such a precise result, because in a streaming environment the memory is limited (relative to the stream length). Therefore, we resort to an approximate result and introduce a summary representation called core-micro-cluster.

DEFINITION 3.3. (*core-micro-cluster*) A core-micro-cluster, abbr. *c-micro-cluster*, at time t is defined as $CMC(w, c, r)$ for a group of close points p_{i_1}, \dots, p_{i_n} with time stamps T_{i_1}, \dots, T_{i_n} . $w = \sum_{j=1}^n f(t - T_{i_j})$, $w \geq \mu$, is the weight. $c = \frac{\sum_{j=1}^n f(t - T_{i_j}) p_{i_j}}{w}$ is the center. $r = \frac{\sum_{j=1}^n f(t - T_{i_j}) \text{dist}(p_{i_j}, c)}{w}$, $r \leq \epsilon$, is the radius, where $\text{dist}(p_{i_j}, c)$ denotes the Euclidean distance between point p_{i_j} and the center c .

Notice that the weight of c-micro-clusters must be above or equal to μ and the radius must be below or equal to ϵ . In such a sense, the c-micro-cluster is a ‘‘dense’’ micro-cluster. Because of the constraint on radius, N_c , the number of c-micro-clusters is much larger than the number of natural clusters. On the other hand, it is significantly smaller than the number of points in the data stream due to its constraint on weight. Since each point is uniquely assigned to one of the c-micro-clusters, N_c is below or equal to $\frac{W}{\mu}$. In addition, when

a clustering request arrives, each c-micro-cluster will be labeled to get the final result, as illustrated in Figure 1.

The clusters with arbitrary shape in data streams are described by a set of non-redundant c-micro-clusters. In fact, $\{cmc_1, cmc_2, \dots, cmc_{N_c}\}$, the set of c-micro-clusters are supposed to correspond to the coverage of the *core objects* in the clusters. And all of them are necessary to cover the core objects.

In an evolving data stream, the role of clusters and outliers often exchange, and any c-micro-cluster is formed gradually as the data stream proceeds. Therefore, we introduce the structures of potential c-micro-clusters and outlier-micro-clusters for incremental computation, which are similar to those in [2]. The main differences between them are their different constraints on weight, $w \geq \beta\mu$ and $w < \beta\mu$, respectively.

DEFINITION 3.4. (*potential c-micro-cluster*) A potential c-micro-cluster, abbr. *p-micro-cluster*, at time t for a group of close points p_{i_1}, \dots, p_{i_n} with time stamps T_{i_1}, \dots, T_{i_n} is defined as $\{\overline{CF^1}, \overline{CF^2}, w\}$. $w = \sum_{j=1}^n f(t - T_{i_j})$, $w \geq \beta\mu$, is the weight. β , $0 < \beta \leq 1$, is the parameter to determine the threshold of outlier relative to c-micro-clusters. $\overline{CF^1} = \sum_{j=1}^n f(t - T_{i_j}) p_{i_j}$ is the weighted linear sum of the points. $\overline{CF^2} = \sum_{j=1}^n f(t - T_{i_j}) p_{i_j}^2$ is the weighted squared sum of the points.

The center of p-micro-cluster is $c = \frac{\overline{CF^1}}{w}$. And the radius of p-micro-cluster is $r = \sqrt{\frac{|\overline{CF^2}|}{w} - \left(\frac{|\overline{CF^1}|}{w}\right)^2}$ ($r \leq \epsilon$).

DEFINITION 3.5. (*outlier micro-cluster*) An outlier micro-cluster, abbr. *o-micro-cluster*, at time t for a group of close points p_{i_1}, \dots, p_{i_n} with time stamps T_{i_1}, \dots, T_{i_n} is defined as $\{\overline{CF^1}, \overline{CF^2}, w, t_o\}$. The definitions of w , $\overline{CF^1}$, $\overline{CF^2}$, center and radius are the same as the p-micro-cluster. $t_o = T_{i_1}$ denotes the creation time of the o-micro-cluster, which is used to determine the life span of the o-micro-cluster. However $w < \beta\mu$. That is, because the weight is below the threshold of outlier, the micro-cluster corresponds to outliers.

PROPERTY 3.1. p-micro-clusters and o-micro-clusters can be maintained incrementally.

Proof. Consider a p-micro-cluster $c_p = (\overline{CF^2}, \overline{CF^1}, w)$, if no points are merged by c_p for time interval δt , $c_p = (2^{-\lambda\delta t} \cdot \overline{CF^2}, 2^{-\lambda\delta t} \cdot \overline{CF^1}, 2^{-\lambda\delta t} \cdot w)$. If point p is merged by c_p , $c_p = (\overline{CF^2} + p^2, \overline{CF^1} + p, w + 1)$. Similar procedure can prove the property of o-micro-clusters.

4 Clustering Algorithm

Our clustering algorithm can be divided into two parts: (1) online part of micro-cluster maintenance, (2) offline

Algorithm 1 Merging (p)

```
1: Try to merge  $p$  into its nearest p-micro-cluster  $c_p$ ;  
2: if  $r_p$  (the new radius of  $c_p$ )  $\leq \epsilon$  then  
3:   Merge  $p$  into  $c_p$ ;  
4: else  
5:   Try to merge  $p$  into its nearest o-micro-cluster  $c_o$ ;  
6:   if  $r_o$  (the new radius of  $c_o$ )  $\leq \epsilon$  then  
7:     Merge  $p$  into  $c_o$ ;  
8:     if  $w$  (the new weight of  $c_o$ )  $> \beta\mu$  then  
9:       Remove  $c_o$  from outlier-buffer and create a  
       new p-micro-cluster by  $c_o$ ;  
10:    end if  
11:  else  
12:    Create a new o-micro-cluster by  $p$  and insert it  
    into the outlier-buffer;  
13:  end if  
14: end if
```

part of generating the final clusters, on demand by the user.

4.1 Micro-cluster Maintenance In order to discover the clusters in an evolving data stream, we maintain a group of p-micro-clusters and o-micro-clusters in an online way. All the o-micro-clusters are maintained in a separate memory space, say an *outlier-buffer*. This is based on the observation that most of the new points belong to existing clusters, and therefore can be absorbed by existing p-micro-clusters.

When a new point p arrives, the procedure of merging is described below (see Algorithm 1 for detail).

1. At first, we try to merge p into its nearest p-micro-cluster c_p . If r_p , the new radius of c_p , is below or equal to ϵ , merge p into c_p . This could be achieved by the incremental property of p-micro-clusters.
2. Else, we try to merge p into its nearest o-micro-cluster c_o . If r_o , the new radius of c_o , is below or equal to ϵ , merge p into c_o . And then, we check w the new weight of c_o . If w is above $\beta\mu$, it means that c_o has grown into a potential c-micro-cluster. Therefore, we remove c_o from the outlier-buffer and create a new p-micro-cluster by c_o .
3. Otherwise we create a new o-micro-cluster c_o by p and insert c_o into the outlier-buffer. This is because p does not naturally fit into any existing micro-cluster. The p may be an outlier or the seed of a new micro-cluster.

For each existing p-micro-cluster c_p , if no new point is merged into it, the weight of c_p will decay gradually. If the weight is below $\beta\mu$, it means that c_p becomes an

outlier, and it should be deleted and its memory space released for new p-micro-clusters. Thus, we need to check the weight of each p-micro-cluster periodically. The problem becomes how to determine the periods of checking. In fact, it does not need to check too frequently. Because the minimal time span for a p-micro-cluster fading into an outlier is

$$(4.1) \quad T_p = \lceil \frac{1}{\lambda} \log\left(\frac{\beta\mu}{\beta\mu - 1}\right) \rceil,$$

which is determined by the equation $2^{-\lambda T_p} \beta\mu + 1 = \beta\mu$. Therefore, we check each p-micro-cluster every T_p time periods. This checking strategy ensures that the maximal number of p-micro-clusters in memory is $\frac{W}{\beta\mu}$, as the overall weight of data streams is a constant W .

The problem is that the number of o-micro-clusters may continuously increase as data streams proceed. It becomes worse when a lot of outliers exist. On the other hand, we need to keep the o-micro-clusters which will grow into p-micro-clusters, because at the initial stage of any new cluster its weight is relatively small compared with existing clusters. So we should provide opportunity for an o-micro-cluster to grow into a p-micro-cluster, while promptly getting rid of the o-micro-cluster which is a *real* outlier. Ideally, we should wait infinite time to determine whether an o-micro-cluster could become a p-micro-cluster or not. However, this strategy will cost a lot of memory. Therefore, we introduce an approximate method to distinguish these two types of o-micro-clusters, and periodically prune the “real” outlier micro-clusters in the outlier-buffer.

It is natural to check each o-micro-cluster every T_p time periods. At these time points, we compare the weight of each o-micro-cluster with its lower limit of weight (denoted as ξ). If the weight of an o-micro-cluster is below its lower limit of weight, that means the o-micro-cluster may not grow into a p-micro-cluster from current aspect. And we can safely delete it from the outlier-buffer. The lower limit of weight is defined as

$$(4.2) \quad \xi(t_c, t_o) = \frac{2^{-\lambda(t_c - t_o + T_p)} - 1}{2^{-\lambda T_p} - 1},$$

which is a function of t_c (i.e., current time) and t_o (i.e., the creation time of the o-micro-cluster). t_o is maintained in the t field of the o-micro-cluster. When $t_c = t_o$, i.e., at the creation time of the o-micro-cluster, $\xi = 1$. As time elapses, ξ increases and $\lim_{t_c \rightarrow \infty} \xi(t_c) = \frac{1}{1 - 2^{-\lambda T_p}} = \beta\mu$. That is, the longer an o-micro-cluster exists, the larger its weight is expected to be. The detailed procedure is described in Algorithm 2.

Obviously, the strategy of pruning o-micro-clusters may introduce some error on the weights of o-micro-clusters and p-micro-clusters. Fortunately, we can

Algorithm 2 DenStream ($DS, \epsilon, \beta, \mu, \lambda$)

```
1:  $T_p = \lceil \frac{1}{\lambda} \log(\frac{\beta\mu}{\beta\mu-1}) \rceil$ ;
2: Get the next point  $p$  at current time  $t$  from data
   stream  $DS$ ;
3: Merging( $p$ );
4: if  $(t \bmod T_p) = 0$  then
5:   for each p-micro-cluster  $c_p$  do
6:     if  $w_p$  (the weight of  $c_p$ )  $< \beta\mu$  then
7:       Delete  $c_p$ ;
8:     end if
9:   end for
10: for each o-micro-cluster  $c_o$  do
11:    $\xi = \frac{2^{-\lambda(t-t_o+T_p)}-1}{2^{-\lambda T_p}-1}$ ;
12:   if  $w_o$  (the weight of  $c_o$ )  $< \xi$  then
13:     Delete  $c_o$ ;
14:   end if
15: end for
16: end if
17: if a clustering request arrives then
18:   Generating clusters;
19: end if
```

guarantee that for any p-micro-cluster c_p , if the current exact weight of c_p (the weight of all data stream points in the radius of c_p , i.e., without summarizing) is above $\beta\mu$, it must exist in the outlier-buffer or the group of p-micro-clusters. And if the current exact weight of c_p is above $2\beta\mu$, it must exist in the group of p-micro-clusters.

Let w_e denote the exact weight of c_o (or c_p), w denote the weight maintained by c_o (or c_p), t_o denote the creation time of c_o (or c_p), and T denote the elapsed time from the very beginning of the data stream:

LEMMA 4.1. *Whenever an o-micro-cluster c_o is pruned, $w_e \leq \beta\mu$.*

Proof. Depend on the existence of micro-clusters located in c_o before its creation or not, there are two cases:

1. If there is no micro-clusters located in c_o before, $w_e = w$. According to the pruning rule, $w_e < \frac{2^{-\lambda(t_c-t_o+T_p)}-1}{2^{-\lambda T_p}-1} < \beta\mu$;
2. If there is micro-cluster c_x located in c_o before, according to the pruning rule, the maximal weight of c_x is $\beta\mu$ when it is pruned. Therefore, $w_e < w + 2^{-\lambda(t_c-t_o+T_p)}\beta\mu \leq \frac{2^{-\lambda(t_c-t_o+T_p)}-1}{2^{-\lambda T_p}-1} + 2^{-\lambda(t_c-t_o+T_p)}\beta\mu \leq \beta\mu$.

In these two cases, we get that $w_e \leq \beta\mu$ whenever c_o is deleted.

LEMMA 4.2. *For any p-micro-cluster c_p , $w \leq w_e \leq w + 2^{-\lambda(t_c-t_o)}\beta\mu$.*

Proof. If c_p is created at the very beginning of data stream, $w = w_e$. Otherwise there are maybe some micro-clusters located in c_p before the creation of c_p . From Lemma 4.1, we infer that the exact weight of c_p is at most $\beta\mu$ when the last pruning took place, and it fades into $2^{-\lambda(t_c-t_o)}\beta\mu$ currently. Therefore, $w_e \leq w + 2^{-\lambda(t_c-t_o)}\beta\mu$ holds.

THEOREM 4.1. *Algorithm DenStream maintains at most $\frac{v}{\lambda} \log(\frac{\beta\mu}{\beta\mu-1})(\log(T) - \log(\frac{1}{\lambda} \log(\frac{\beta\mu}{\beta\mu-1})))$ o-micro-clusters.*

Proof. We divide the time elapsed into buckets of width $l = \lceil \frac{1}{\lambda} \log(\frac{\beta\mu}{\beta\mu-1}) \rceil$. Let B be the current bucket id and c_i ($1 \leq i \leq B$) denote the number of o-micro-clusters in current group of o-micro-clusters C , which are created in bucket $B - i + 1$. The number of points merged by such an o-micro-cluster is at least i from bucket $B - i + 1$ to B ; otherwise it would have been deleted. Let v be the number of points arrived in unit time, i.e., stream speed. The number of points arrived in each bucket is vl . We get the following constraints:

$$(4.3) \quad \sum_{i=1}^j ic_i \leq jvl \quad \text{for } j = 1, 2, \dots, B.$$

From Inequality 4.3, it could be proven by induction that,

$$(4.4) \quad \sum_{i=1}^j c_i \leq \sum_{i=1}^j \frac{vl}{i} \quad \text{for } j = 1, 2, \dots, B.$$

Since $|C| = \sum_{i=1}^j c_i$, from Inequality 4.4, we get

$$|C| \leq \sum_{i=1}^B \frac{vl}{i} \leq vl \log B = \frac{v}{\lambda} \log(\frac{\beta\mu}{\beta\mu-1})(\log(T) - \log(\frac{1}{\lambda} \log(\frac{\beta\mu}{\beta\mu-1})))$$

Theorem 4.1 shows that the total number of micro-clusters (including the p-micro-clusters and o-micro-clusters) increases logarithmically with increasing stream length. However, we claim that the total number of micro-clusters in our algorithm is a small integer in real applications. For example, let unit time be 1 millisecond, $\beta\mu = 2$, $v = 1000$ and $\lambda = 0.25$, the total number of micro-clusters for 1 million years would be less than 512K.

Initialization We apply the DBSCAN algorithm to the first $InitN$ points $\{P\}$ to initialize the online

process. We initialize a group of p-micro-clusters by scanning $\{P\}$. For each point p in $\{P\}$, if the total weight (i.e., the number of points) in its ϵ neighborhood is above $\beta\mu$, then we create a p-micro-cluster by p and its neighbors, and delete them from $\{P\}$.

4.2 Generating Clusters The on-line maintained micro-clusters capture the density area of data streams. However, in order to get meaningful clusters, we need to apply some clustering algorithm to get the final result. When a clustering request arrives, a variant of DBSCAN algorithm is applied on the set of on-line maintained p-micro-clusters to get the final result of clustering. Each p-micro-cluster c_p is regarded as a virtual point located at the center of c_p with weight w .

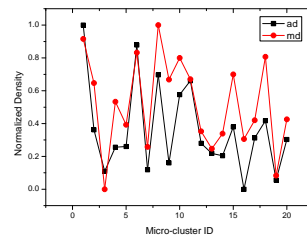
The variant of DBSCAN algorithm includes two parameters ϵ and μ . We adopt the concept of density-connectivity, similar to [8], to determine the final clusters. That is, all the density-connected p-micro-clusters form a cluster.

DEFINITION 4.1. (directly density-reachable) A p-micro-cluster c_p is directly density-reachable from a p-micro-cluster c_q wrt. ϵ and μ , if the weight of c_q is above μ (i.e., c_q corresponds a c-micro-cluster) and $dist(c_p, c_q) \leq 2 \cdot \epsilon$, where $dist(c_p, c_q)$ is the distance between the centers of c_p and c_q .

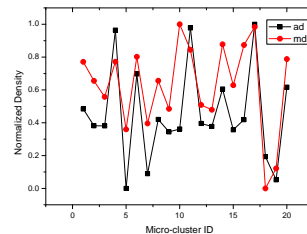
Intuitively, since the points are distributed within the radiuses of the micro-clusters, we regard two p-micro-clusters as density-reachable when they are tangent (or intersecting), i.e., $dist(c_p, c_q) \leq 2 \cdot \epsilon$. The two micro-clusters may actually not intersect even if their distance is smaller than $2 \cdot \epsilon$, because the actual radiuses of the micro-clusters can be smaller than ϵ . In such a case, they are not directly density-reachable. Fortunately, we can detect this phenomena by further checking whether $dist(c_p, c_q) \leq r_p + r_q$ or not, where r_p and r_q are the radiuses of c_p and c_q , respectively. And the final result will be similar to the one which is produced by applying DBSCAN to the original data stream with parameter ϵ and μ .

DEFINITION 4.2. (density-reachable) A p-micro-cluster c_p is density-reachable from a p-micro-cluster c_q wrt. ϵ and μ , if there is a chain of p-micro-clusters c_{p_1}, \dots, c_{p_n} , $c_{p_1} = c_q$, $c_{p_n} = c_p$ such that $c_{p_{i+1}}$ is directly density-reachable from c_{p_i} .

DEFINITION 4.3. (density-connected) A p-micro-cluster c_p is density-connected to a p-micro-cluster c_q wrt. ϵ and μ if there is a p-micro-cluster c_m such that both c_p and c_q are density-reachable from c_m wrt. ϵ and μ .



(a) stream length=30,000



(b) stream length=60,000

Figure 2: ad vs. md

5 Discussion

The density is maintained by the weight of the micro-cluster with limited radius. There are two alternative definitions of the radius. The one is r_{max} , which is the maximal distance from any point in micro-cluster c_p to the center of c_p . The density corresponding to r_{max} is denoted as md . The other is r_{avg} , which is the average distance from the points in c_p to the center of c_p . The density corresponding to r_{avg} is denoted as ad . In static environment, r_{max} is a more intuitive measurement. However, it is hard to get r_{max} precisely with limited memory in a streaming environment, because the points in c_p constantly fade out. In the worst case, if the distance from each point to the center is monotonically decreasing, it costs $O(n)$ space. In the random case, the expected space cost is $O(\log n)$. However, it makes the assumption on the arrival of data and introduces additional computation and memory cost for each micro-cluster.

We claim that for density measurement in a data stream, ad is simple but effective. To study the effectiveness of ad , we download the SEQUOIA 2000 benchmark data [16]. There are four types of data in the database: raster data, point data, polygon data and directed graph data. The point data set contains 62,584 Californian names of landmarks. This data set is converted into a data stream by taking the data input order as the order of streaming. A set of experiments is designed to compare ad to md . Figures 2(a) and (b) depict the comparison result on the stream with length 30,000 and 60,000, respectively. We compute ad for 20 randomly selected micro-clusters. The corresponding

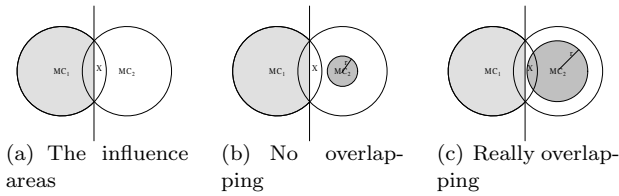


Figure 3: The overlapping problem

md is gotten by checking the number of points in the ϵ neighborhood of the center of the micro-cluster in the stream. We normalize the density as follows:

$$Normalized\ Den = \frac{den - minimum\ den}{maximum\ den - minimum\ den}$$

Figure 2 shows that the trend of ad curve is very similar to that of md . Since we are looking for a density measurement to determine whether two micro-clusters are similar (or in the same cluster), we believe that ad is a sufficiently good replacement for md .

P-micro-clusters may have overlapping. Figure 3(a) illustrates the influence areas of two overlapping micro-clusters. The light grey area is the influence area of MC_1 , while the white area is the influence area of MC_2 , because each new point is merged into its nearest micro-cluster. The border is a hyperplane which is perpendicular to the line between MC_1 and MC_2 and divides the line into exactly two halves. Obviously, if MC_2 does not exist, the points in the right half of the intersection area of MC_1 and MC_2 (called, X area) should be absorbed by MC_1 . When MC_2 is pruned, the points in MC_2 located in the X area should be reassigned to MC_1 , because from now on the X area belongs to the influence area of MC_1 .

The problem becomes how to estimate the number of points in MC_2 located in the X area. One strategy is assuming that the points in MC_2 follow normal distribution and estimating a pseudo-point to represent the points in MC_2 located in the X area. However, the accuracy of this strategy is not very good. Therefore, when MC_2 is pruned, we calculate the radius of MC_2 , and check whether the area in the radius of MC_2 overlaps with the X area. If there is no overlapping, as shown in Figure 3(b), we needn't do the reassignment. Otherwise, as shown in Figure 3(c), we estimate a pseudo-point which represents the points in MC_2 located in the X area, by assuming the points in MC_2 follow a gaussian distribution, and reassign this pseudo-point to MC_1 . The mean of the distribution equals to the center of MC_2 and the variance of the distribution equals to the radius of MC_2 .

6 Experimental Evaluation

In this section, we present an experimental evaluation of DenStream. We implemented DenStream as well as the comparative method CluStream in Microsoft Visual C++. All experiments were conducted on a 3.4 GHz PentiumIV PC with 1GB memory, running Microsoft Windows Xp.

6.1 Data Sets and Evaluation To evaluate the clustering quality, scalability, and sensitivity of the DenStream algorithm both real and synthetic data sets are used. The three synthetic data sets, DS1, DS2 and DS3, are depicted in Figures 4(a), (b) and (c), respectively, which are the similar data sets used by [8]. Each of them contains 10,000 points. We generate an evolving data stream (denoted as EDS) by randomly choosing one of the data sets (DS1, DS2 and DS3) 10 times, for each time the chosen data set forms a 10,000 points segment of the data stream, and the total length of the evolving data stream is 100,000. Because we know the true (class) cluster labels of each points in the data sets DS1, DS2 and DS3, we can get the true cluster (class) label of each points in EDS. The real data sets are the KDD CUP'99 Network Intrusion Detection data set (all 34 continuous attributes out of the total 42 available attributes are used) and KDD CUP'98 Charitable Donation data set (as in [1], the total 56 fields extracted from 481 fields of each record are used). Both of them have been used in [1]. And they are converted into data streams by taking the data input order as the order of streaming. The clustering quality is evaluated by the average purity of clusters which is defined as follows:

$$pur = \frac{\sum_{i=1}^K \frac{|C_i^d|}{|C_i|}}{K} \times 100\%,$$

where K denotes the number of clusters. $|C_i^d|$ denotes the number of points with the dominant class label in cluster i . $|C_i|$ denotes the number of points in cluster i .

Intuitively, the purity measures the purity of the clusters with respect to the true cluster (class) labels that are known for our data sets. Since the weight of points fades out gradually, we compute the purity by only the points arriving in a pre-defined horizon H (or window) from current time. Our experimental results show that the purity results are insensitive to the horizon.

In order to test the scalability of DenStream, we also generate some synthetic data sets with different numbers of dimensions and numbers of natural clusters. As in [1], the points of each synthetic data set follow a series of Gaussian distributions, while the mean and variance of current distribution are changed for every

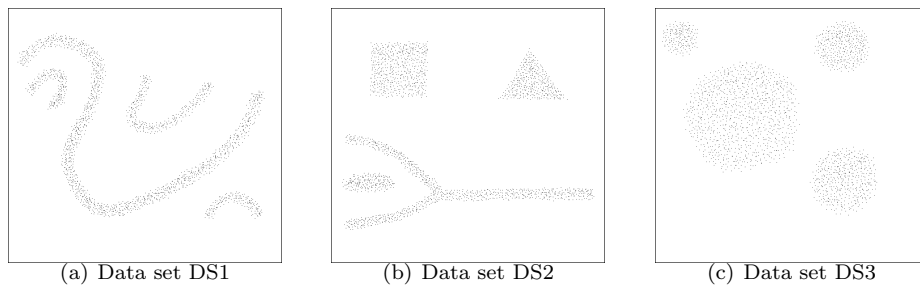


Figure 4: Synthetic data sets

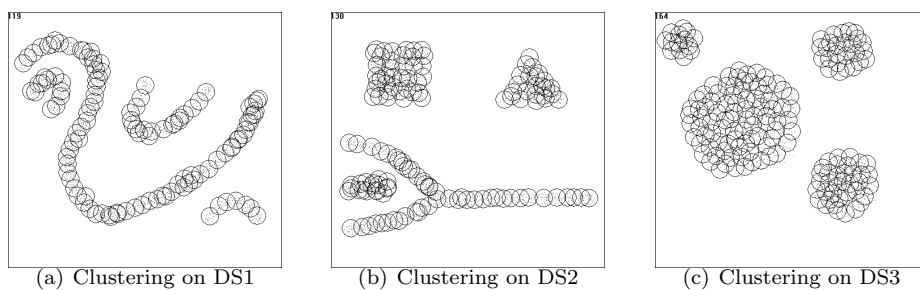


Figure 5: Clustering on DS1, DS2 and DS3

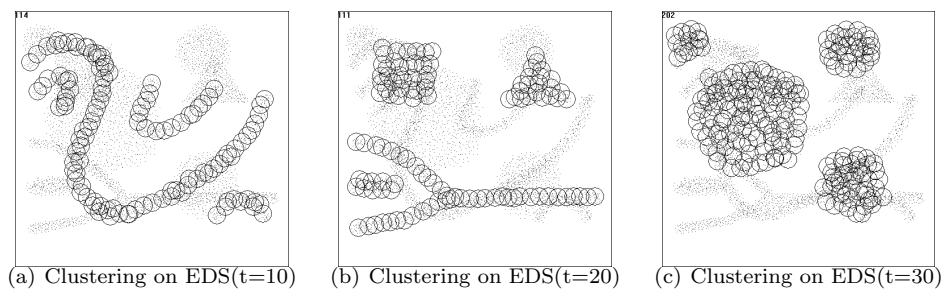


Figure 6: Clustering on the evolving data stream EDS

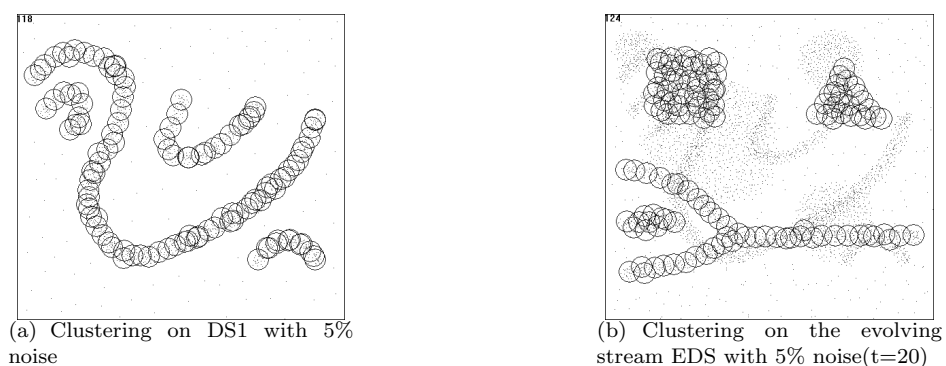


Figure 7: Clustering on data streams with noise

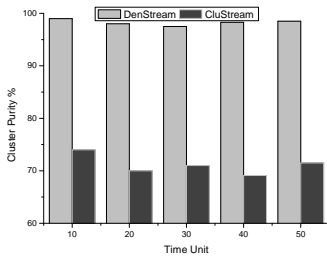


Figure 8: Clustering quality(EDS data stream, horizon=2, stream speed=2000)

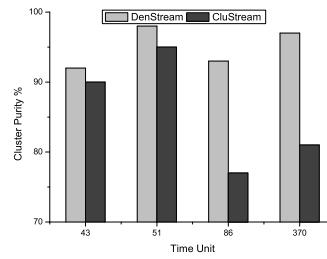


Figure 10: Clustering quality(Network Intrusion data set, horizon=1, stream speed=1000)

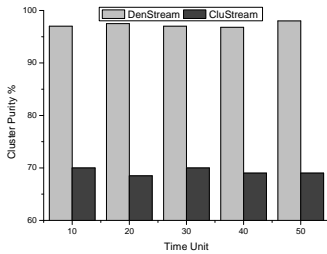


Figure 9: Clustering quality(EDS data stream, horizon=10, stream speed=1000)

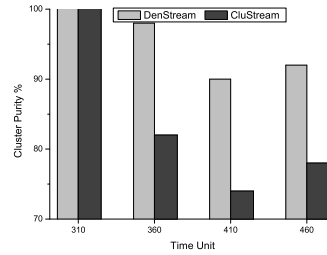


Figure 11: Clustering quality(Network Intrusion data set, horizon=5, stream speed=1000)

10,000 points during the synthetic data generation. We adopt the following notations to characterize the synthetic data sets: ‘B’ indicates the number of data points in the data set, whereas ‘C’ and ‘D’ indicate the number of natural clusters, the dimensionality of each point, respectively. For example, B400C5D20 means the data set contains 400,000 data points of 20-dimensions, belonging to 5 different clusters.

The input parameters of the DenStream algorithm are discussed in the following:

1. For ϵ , if it is too large, it may mess up different clusters. If it is too small, it requires a corresponding smaller μ . However, a smaller μ will result in a larger number of micro-clusters. Therefore, we apply DBSCAN on the initial points. We set $\epsilon = \alpha \cdot d_{min}$, where α ($0 < \alpha < 1$) is fixed by the requirement of precision in real applications, and d_{min} is the minimal distance between the nearest pair of points in different clusters.
2. After the setting of ϵ , μ can be heuristically set by the average number of points in the ϵ neighborhood of each points in each clusters. When the density of data streams is greatly varying, it’s relatively hard to choose proper parameters. Generally, we should choose a smallest acceptable μ .
3. λ is chosen by the requirement of real applications. β can be estimated by the character of data

streams. We will test the sensitivity of clustering quality to parameters λ and β in Section 6.4.

Unless particularly mentioned, the parameters of DenStream adopt the following setting: initial number of points $InitN = 1000$, stream speed $v = 1000$, decay factor $\lambda = 0.25$, $\epsilon = 16$, $\mu = 10$, outlier threshold $\beta = 0.2$. The parameters for CluStream are chosen to be the same as those adopted in [1].

6.2 Clustering Quality Evaluation At first, we test the clustering quality of DenStream. Figures 5(a), (b) and (c) show the results from three non-evolving data streams DS1, DS2 and DS3, respectively, while Figures 6(a), (b) and (c) show the results from the evolving data stream EDS at different times. The first portion of points in the EDS are DS1, then DS2, DS3... In the figures, the points denote the raw data, the circles indicate the micro-clusters, and the number in the top-left corner is the number of micro-clusters. It can be seen that DenStream precisely captures the shape of each cluster in all the data streams.

Figure 8 shows the purity results of DenStream and CluStream in a small horizon on the EDS data stream. The stream speed v is set at 2000 points per time unit and horizon $H = 2$. It can be seen that DenStream has a very good clustering quality. Its clustering purity is always higher than 95% and much better than CluStream whose purity is always about

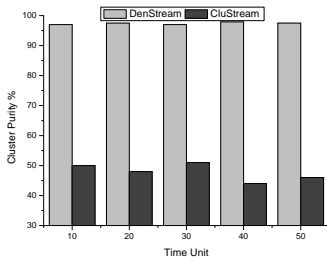


Figure 12: Clustering quality(EDS data stream with 1% noise, horizon=2, stream speed=2000)

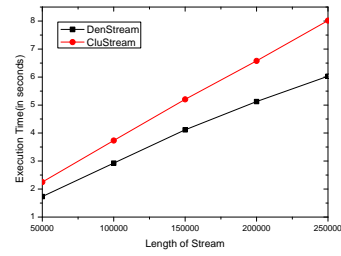


Figure 14: Execution time vs. length of stream(Network Intrusion data set)

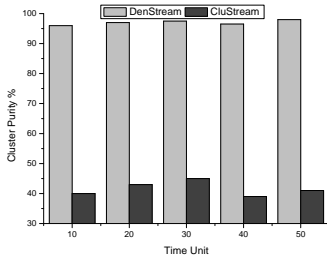


Figure 13: Clustering quality(EDS data stream with 5% noise, horizon=10, stream speed=1000)

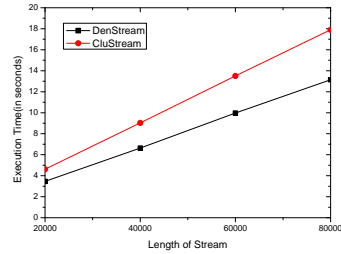


Figure 15: Execution time vs. length of stream(Charitable Donation data set)

70%. For example, at time 30, DenStream groups different natural clusters into different clusters, while CluStream groups two natural clusters into one cluster, thus, the purity of DenStream is 25% higher than CluStream. We also set the stream speed v at 1000 points per time unit and horizon $H = 10$ for EDS. Figure 9 shows similar results as Figure 8. We conclude that DenStream can also get much higher clustering quality than CluStream in a relatively large horizon.

We also compare DenStream with CluStream on the Network Intrusion data set. Figure 10 shows the results in a small horizon $H = 1$. We test the data set at selected time points when some attacks happen. For example, at time 370 there are 43 “portsweep” attacks, 20 “pod” attacks, and 927 “neptune” attacks in $H = 1$. It can be seen that DenStream clearly outperforms CluStream and the purity of DenStream is always above 90%. For example, at time 86 the purity of DenStream is about 94% and 20% higher than that of CluStream. This is because DenStream can separate different attacks into different clusters, while CluStream may merge different attacks into one attack (or normal connections). Figure 11 shows that DenStream also outperforms CluStream in a relatively large horizon $H = 5$ at most of the times expect time 310. We checked the data set and found that at that time all the points in horizon $H = 5$ belong to one “smurf” attack, that is, any algorithm can achieve 100% purity at that time including CluStream.

6.2.1 When Noise Exists There is often some noise in stream applications, which can not fit in any clusters. Therefore, we also evaluate the DenStream algorithm in noisy environments. In order to simulate the noise environment, we add some uniformly distributed random noise in DS1, DS2, DS3 and EDS. Figures 7(a) and (b) show the results from DS1 with 5% random noise and EDS with 5% random noise, respectively. Both of them demonstrate that DenStream can effectively capture the shape of clusters while remove the noise in the data streams.

We also calculate the purity result of DenStream compared to CluStream over EDS with noise. Figures 12 and 13 show the clustering purity results of EDS data streams with 1% and 5% noise, respectively. Both of them demonstrate that our DenStream algorithm can also achieve very high clustering quality when noise exists. On the contrary, the clustering quality of CluStream is not very good in the noisy environment. For example, for EDS data stream with 5% noise, the purity of DenStream is about 96% at time 20, while the purity of CluStream is just about 42%, much lower than that of DenStream. The high quality of DenStream benefits from its effective pruning strategy, which promptly gets rid of the outliers while keeps the potential clusters. On the contrary, CluStream lacks an effective mechanism to distinguish these two types of new micro-clusters. Therefore, it wastes a lot of memory to keep the outliers and merges or deletes a lot of micro-

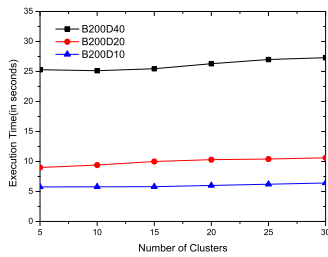


Figure 16: Execution time vs. number of clusters

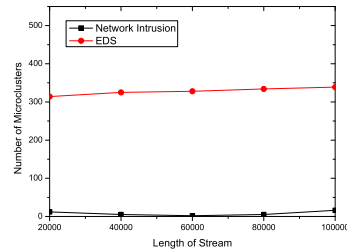


Figure 18: Memory usage

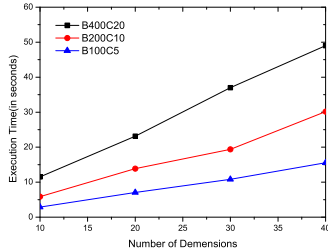


Figure 17: Execution time vs. dimensionality

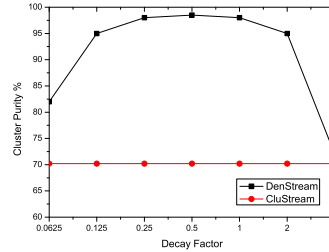


Figure 19: Clustering quality vs. decay factor λ

clusters which correspond to natural clusters.

6.3 Scalability Results

Execution Time The efficiency of algorithms is measured by the execution time. The CluStream algorithm needs to periodically store the current snapshot of micro-clusters. And the snapshots are maintained in disk in [1]. In order to improve the efficiency of CluStream, we store the snapshots in memory in our implementation.

We use both Network Intrusion Detection and Charitable Donation data sets to test the efficiency of DenStream against CluStream. Figure 14 shows the execution time for the Network Intrusion data set. We can see that both the execution time of DenStream and CluStream grow linearly as the stream proceeds, and DenStream is more efficient than CluStream. In addition, DenStream takes less than 3 seconds to process 100,000 points. Thus, DenStream can comfortably handle high speed data streams. Figure 15 shows that DenStream is also more efficient than CluStream for the Charitable Donation data set.

The execution time of DenStream is then evaluated on data streams with various dimensionality and different numbers of natural clusters. Synthetic data sets are used for these evaluations, because any combination of dimensionality and number of natural clusters could be gotten in the generation of data sets.

The first series of data sets were generated by vary-

ing the number of natural clusters from 5 to 30, while fixing the number of points and dimensionality. Figure 16 demonstrates that DenStream is of linear execution time in proportion to the number of natural clusters. It can also be seen that the execution time grows very slowly, because the number of c -micro-clusters stays almost unchanged as the number of natural clusters increases. For example, for data set series B200D40, when the number of clusters changes from 5 to 30, the execution time only increases by 2.5 seconds.

The other three series of data sets were generated by varying the dimensionality from 10 to 40, while fixing the number of points and natural clusters. Figure 17 shows that as the dimensionality increases, the execution time increases linearly.

Memory Usage We use both Network Intrusion Detection and EDS data stream to evaluate the memory usage of DenStream. The memory usage is measured by the number of micro-clusters. Figure 18 shows that the memory usage of DenStream is limited and bounded as the streams proceed. For example, for EDS data stream, when the stream length changes from 80,000 to 100,000, the memory usage only increases by 5.

6.4 Sensitivity Analysis An important parameter of DenStream is the decay factor λ . It controls the importance of historical data to current clusters. In our previous experiments, we set it to 0.25, which is a moderate setting. We also test the clustering quality

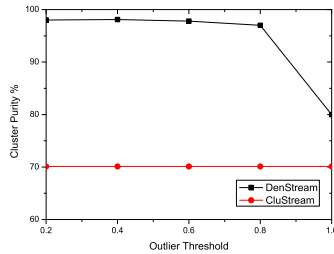


Figure 20: Clustering quality vs. outlier threshold β

by varying λ from 0.00625 to 4. Figure 19 shows the results. When λ is set to a relatively small or high value, the clustering quality becomes poor. For example, when $\lambda = 0.00625$, the purity is about 82%. When $\lambda = 4$, the points decays soon after their arrival, and only a small amount of recent points contributes to the final result. So the result is also not very good. However, the quality of DenStream is still higher than that of CluStream. It can be seen that if λ ranges from 0.125 to 1, the clustering quality is quite good and stable, and always above 95%.

Another important parameter is the outlier threshold β . Figure 20 shows the clustering quality of DenStream when β is varying from 0.2 to 1. If β ranges between 0.2 and 0.6, the clustering quality is very good. However, if it is set to a relatively high value like 1, the quality deteriorates greatly. Because a lot of points corresponding to potential clusters are pruned, the quality is reduced. Note again, that DenStream outperforms CluStream for all parameter settings.

7 Conclusion

In this paper, we have proposed DenStream, an effective and efficient method for clustering an evolving data stream. The method can discover clusters of arbitrary shape in data streams, and it is insensitive to noise. The structures of p-micro-clusters and o-micro-clusters maintain sufficient information for clustering, and a novel pruning strategy is designed to limit the memory consumption with precision guarantee. Our experimental performance evaluation over a number of real and synthetic data sets demonstrates the effectiveness and efficiency of DenStream in discovering clusters of arbitrary shape in data streams.

The future work includes the following topics: the discovery of clusters with arbitrary shape at multiple levels of granularity, dynamic adaption of the parameters in data streams, and investigation of our framework for outlier detection and density-based clustering in other stream models, in particular, in a sliding window model.

References

- [1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proc. of VLDB*, 2003.
- [2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. In *Proc. of VLDB*, 2004.
- [3] M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *Proc. of SIGMOD*, pages 49–60, 1999.
- [4] M. Charikar, L. O’Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proc. of STOC*, pages 30–39, 2003.
- [5] B. Dai, J. Huang, M. Yeh, and M. Chen. Clustering on demand for multiple data streams. In *Proc. of ICDM*, pages 367–370, 2004.
- [6] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. of KDD*, 2000.
- [7] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proc. of the 18th International Conference on Machine Learning (ICML 2001)*, pages 106–113, 2001.
- [8] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of KDD*, 1996.
- [9] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proc. of VLDB*, pages 323–333, 1998.
- [10] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O’Callaghan. Clustering data streams: theory and practice. In *IEEE Transactions on Knowledge and Data Engineering*, pages 515–528, 2003.
- [11] S. Guha, N. Mishra, R. Motwani, and L. O’Callaghan. Clustering data stream. In *Proc. of FOCS*, 2000.
- [12] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *Proc. of SIGMOD*, 1998.
- [13] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proc. of KDD*, pages 58–65, 1998.
- [14] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. of VLDB*, pages 346–357, 2002.
- [15] O. Nasraoui, C. Cardona, C. Rojas, and F. Gonzalez. Tecno-streams: tracking evolving clusters in noisy data streams with a scalable immune system learning model. In *Proc. of ICDM*, pages 235–242, 2003.
- [16] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The sequoia 2000 storage benchmark. In *Proc. of SIGMOD*, pages 2–11, 1993.
- [17] W. Wang, J. Yang, and R. R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *Proc. of VLDB*, pages 186–195, 1997.
- [18] J. Yang. Dynamic clustering of evolving streams with a single pass. In *Proc. of ICDE*, 2003.