Data Mining and Knowledge Discovery, 10, 251–273, 2005


# On High Dimensional Projected Clustering of Data Streams

CHARU C. AGGARWAL charu@us.ibm.com
*IBM T. J. Watson Research Center, Yorktown Heights, NY*

JIAWEI HAN hanj@cs.uiuc.edu
*Department of Computer Science, University of Illinois at Urbana Champaign, Urbana IL*

JIANYONG WANG jianyong@cs.umn.edu
*Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China*

PHILIP S. YU psyu@us.ibm.com
*IBM T. J. Watson Research Center, Hawthorne, NY*

**Editor:** Johannes Gehrke

*Received October 1, 2004; Accepted April 1, 2005*

**Abstract.** The data stream problem has been studied extensively in recent years, because of the great ease in collection of stream data. The nature of stream data makes it essential to use algorithms which require only one pass over the data. Recently, single-scan, stream analysis methods have been proposed in this context. However, a lot of stream data is high-dimensional in nature. High-dimensional data is inherently more complex in clustering, classification, and similarity search. Recent research discusses methods for projected clustering over high-dimensional data sets. This method is however difficult to generalize to data streams because of the complexity of the method and the large volume of the data streams.

In this paper, we propose a new, high-dimensional, projected data stream clustering method, called HPStream. The method incorporates a *fading cluster structure*, and the *projection based clustering* methodology. It is incrementally updatable and is highly scalable on both the number of dimensions and the size of the data streams, and it achieves better clustering quality in comparison with the previous stream clustering methods. Our performance study with both real and synthetic data sets demonstrates the efficiency and effectiveness of our proposed framework and implementation methods.

**Keywords:** high dimensional, projected clustering, data streams

## 1. Introduction

The problem of data streams has gained importance in recent years because of advances in hardware technology. These advances have made it easy to store and record numerous transactions and activities in everyday life in an automated way. The ubiquitous presence of data streams in a number of practical domains has generated a lot of research in this area (Babcock et al., 2002; Domingos and Hulten, 2000; Feigenbaum et al., 2000; Guha et al., 2000; O'Callaghan et al., 2002). One of the important problems which has recently

been explored in the data stream domain is that of clustering (O'Callaghan et al., 2002). The clustering problem is especially interesting for the data stream domain because of its application to data summarization and outlier detection.

The clustering problem is defined as follows: *for a given set of data points, we wish to partition them into one or more groups of similar objects, where the notion of similarity is defined by a distance function.* There have been a lot of research work devoted to scalable cluster analysis in recent years (Agrawal et al., 1998; Aggarwal et al., 1999; Guha et al., 1998; Jain and Dubes, 1998; Ng and Han, 1994; Zhang et al., 1996). In the data stream domain, the clustering problem requires a process which can continuously determine the dominant clusters in the data without being dominated by the previous history of the stream.

The high-dimensional case presents a special challenge to clustering algorithms even in the traditional domain of static data sets. This is because of the sparsity of the data in the high-dimensional case. In high-dimensional space, all pairs of points tend to be almost equidistant from one another. As a result, it is often unrealistic to define distance-based clusters in a meaningful way. Some recent work on high-dimensional data uses techniques for *projected clustering* which can determine clusters for a specific subset of dimensions (Aggarwal, 2004; Agrawal et al., 1998; Aggarwal et al., 1999). In these methods, the definitions of the clusters are such that each cluster is specific to a particular group of dimensions. This alleviates the sparsity problem in high-dimensional space to some extent. Even though a cluster may not be meaningfully defined on all the dimensions because of the sparsity of the data, some subsets of the dimensions can always be found on which particular subsets of points form high quality and meaningful clusters. Of course, these subsets of dimensions may vary over the different clusters. Such clusters are referred to as *projected clusters* (Aggarwal et al., 1999).

The concept of a projected cluster is formally defined as follows. Assume that $k$ is the number of clusters to be found. In addition, the algorithm will take as input the dimensionality $l$ of the subspace in which each cluster is reported. The output of the algorithm will be twofold:

- A $(k+1)$-way partition $\{\mathcal{C}_1, \ldots, \mathcal{C}_k, \mathcal{O}\}$ of the data, such that the points in each partition element except the last form a cluster, whereas the points in the last partition element are the *outliers*, which by definition do not cluster well.
- A possibly different set $\mathcal{E}_i$ of dimensions for each cluster $\mathcal{C}_i$, $1 \leq i \leq k$, such that the points in $\mathcal{C}_i$ cluster well in the subspace defined by these vectors. (The vectors for the outlier set $\mathcal{O}$ can be assumed to be the empty set.) For each cluster $\mathcal{C}_i$, the cardinality of the corresponding set $\mathcal{E}_i$ is equal to the user-defined parameter $l$.

In the context of a data stream, the problem of finding projected clusters becomes even more challenging. This is because the additional problem of *finding the relevant set of dimensions for each cluster* makes the problem significantly more computationally intensive in the data stream environment. While the problem of clustering has recently been studied in the data stream environment (Aggarwal et al., 2003; Babcock et al., 2002; Farnstrom et al., 2000), these methods are for the case of full dimensional clustering. In this paper, we will work on the significantly more difficult problem of clustering high-dimensional data stream by

exploring projected clustering methods. We note that existing projected clustering methods such as those discussed in Aggarwal et al. (1999) cannot be easily generalized to the data stream problem because they typically require multiple passes over the data. Furthermore, the algorithms in Aggarwal et al. (1999) are too computationally intensive to be used for the data stream problem. In addition, data streams quickly evolve over time (Aggarwal, 2002, 2003) because of which it is essential to design methods which are designed to effectively adjust with the progression of the stream.

In this paper, we will develop an algorithm for high-dimensional projected stream clustering by continuous refinement of the set of projected dimensions and data points during the progression of the stream. We will refer to this algorithm as HPStream, since it describes the High-dimensional Projected Stream clustering method. The updating of the set of dimensions associated with each cluster is performed in such a way that the points and dimensions associated with each cluster can effectively evolve over time. In order to achieve this goal, we utilize a condensed representation of the statistics of the points inside the clusters known as the *fading cluster structure*. These condensed representations are chosen in such a way that they can be updated effectively in a fast data stream. At the same time, a sufficient amount of statistics is stored so that important measures about the cluster in a given projection can be quickly computed. In the next section, we will discuss the *fading cluster structure* which is useful for such book-keeping. This structure is also capable of performing the updates in such a way that outdated data is temporally discounted. This ensures that in an evolving data stream, the past history is gradually discounted from the computation.

In comparison with the previous literature, we have made substantial progress in the following aspects:

1. HPStream introduces the concept of *projected clustering* to data streams. Since a lot of stream data is high-dimensional in nature, it is necessary to perform high quality high-dimensional clustering. However, the previous stream clustering methods, such as STREAM and CluStream, cannot handle such data well, due to their clustering of data in all the relevant dimensions. Moreover, PROCLUS, though exploring *projected clustering*, cannot handle data streams due to its requirement of multiple scans of the data.
2. HPStream explores a linear update philosophy in projected clustering, achieving both high scalability and high clustering quality. This philosophy was first proposed in BIRCH. CluStream introduces this idea to stream clustering, however, it does not show good quality with high dimensional data. With projected clustering, HPStream can reach consistently high clustering quality due to its adaptability to the nature of real data set, where data shows its tight clustering behavior only at different subsets of dimension combinations.

Besides the above major progress, HPStream has proposed and explored several other innovative ideas. For example, the *fading cluster structure*, nicely integrates historical and current data with a user-specified or user-tunable fading factor. Also, using bit-vector for registration and dynamic update of relevant dimensions, and using minimal radius for clustering quality enhancement have improved the clustering efficiency and accuracy.

The remaining of the paper is organized as follows. In Section 2, we will discuss the basic concepts that are necessary for developing the algorithm. In Section 3, we will introduce the HPStream algorithm of this paper. Section 4 reports our performance study on real and synthetic data sets. We will compare the HPStream algorithm to the full dimensional CluStream algorithm. A brief discussion of the possible extensions of this work is included in Section 5. The conclusions and summary are discussed in Section 6.

## 2. The fading cluster structure: Motivation and concepts

The data stream consists of a set of multi-dimensional records $\bar{X}_1 \ldots \bar{X}_k \ldots$ arriving at time stamps $T_1 \ldots T_k \ldots$. Each data point $\bar{X}_i$ is a multi-dimensional record containing $d$ dimensions, denoted by $\bar{X}_i = (x_i^1 \ldots x_i^d)$. Since the stream clustering process should provide a greater level of importance to recent data points, we introduce the concept of a *fading data structure* which is able to adjust for the recency of the clusters in a flexible way. It is assumed that each data point has a weight defined by a function $f(t)$ to the time $t$. The function $f(t)$ is also referred to as the *fading function*. The value of the fading function lies in the range (0, 1). It is also assumed that the fading function is a monotonic decreasing function which decays uniformly with time $t$. In particular, we choose an exponential form for the fading function. The exponentially fading function is widely used in temporal applications in which it is desirable to gradually discount the history of past behavior. In order to formalize the concept of the fading function, we will define the *half-life* of a point in the data stream.

*Definition 2.1.* The *half life* $t_0$ of a point is defined as the time at which $f(t_0) = (1/2) f(0)$.

Conceptually, the aim of defining a half life is to define the rate of decay of the weight assigned to each data point in the stream. Correspondingly, the *decay-rate* is defined as the inverse of the half life of the data stream. We denote the decay rate by $\lambda = 1/t_0$. In order for the half-life property to hold, we define the *weight* of each point in the data stream by $f(t) = 2^{-\lambda \cdot t}$. From the perspective of the clustering process, the weight of each data point is $f(t)$. It is easy to see that this decay function creates a half life of $1/\lambda$. It is also evident that by changing the value of $\lambda$, it is possible to change the rate at which the importance of the historical information in the data stream decays. The higher the value of $\lambda$, the lower the importance of the historical information compared to more recent data.

We will now define the *fading cluster structure*, a data structure which is designed to capture key statistical characteristics of the clusters generated during the course of a data stream. The aim of the fading cluster structure is to capture a sufficient number of the underlying statistics so that it is possible to compute key characteristics of the underlying clusters.

*Definition 2.2.* A *fading cluster structure* at time t for a set of d-dimensional points $\mathcal{C} = \{X_{i_1} \ldots X_{i_n}\}$ with time stamps $T_{i_1} \ldots T_{i_n}$ is defined as the $(2 \cdot d + 1)$ tuple $\mathcal{FC}(\mathcal{C}, t) = (\overline{FC2^x(\mathcal{C}, t)}, \overline{FC1^x(\mathcal{C}, t)}, W(t))$. The vectors $\overline{FC2^x(\mathcal{C}, t)}$ and $\overline{FC1^x(\mathcal{C}, t)}$ each contain d entries. We will now explain the significance of each of these sets of entries:

1. For each dimension $j$, the jth entry of $\overline{FC2^x(\mathcal{C}, t)}$ is given by the weighted sum of the squares of the corresponding data values in that dimension. The weight of each data point is defined by its level of staleness since its arrival in the data stream. Thus, $\overline{FC2^x(\mathcal{C}, t)}$ contains $d$ values. The $j$-th entry of $\overline{FC2^x(\mathcal{C}, t)}$ is equal to $\sum_{k=1}^{n} f(t - T_{i_k}) \cdot (x_{i_k}^j)^2$. Thus, this entry defines a *time-decaying* second order moment of the data points.
2. For each dimension j, the $j$th entry of $\overline{FC1^x(\mathcal{C}, t)}$ is given by the weighted sum of the corresponding data values. The weight of each data point is defined by its level of staleness since its arrival in the data stream. Thus, $\overline{FC1^x(\mathcal{C}, t)}$ contains d values. The $j$-th entry of $\overline{FC1^x(\mathcal{C}, t)}$ is equal to $\sum_{k=1}^{n} f(t - T_{i_k}) \cdot (x_{i_k}^j)$. Thus, this entry defines a *time-decaying* first order moment of the data points.
3. We also maintain a single entry $W(t)$ containing the sum of all the weights of the data points at time t. Thus, this entry is equal to $W(t) = \sum_{k=1}^{n} f(t - T_{i_k})$. Thus, this entry defines a time-decaying zeroth order moment of the data points.

We note that the above definition essentially contains second-order, first-order and zeroth order moments as in the CluStream algorithm. The only difference is the additional weights added to the cluster feature vectors in order to facilitate time-decay. The clustering structure discussed above satisfies a number of interesting properties. These properties are referred to as *additivity* and *temporal multiplicity*. The additivity property is defined as follows:

*Observation 2.1.* Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two clusters with cluster structures $\mathcal{FC}(\mathcal{C}_1, t)$ and $\mathcal{FC}(\mathcal{C}_2, t)$ respectively. Then, the cluster structure of $\mathcal{C}_1 \cup \mathcal{C}_2$ is given by $\mathcal{FC}(\mathcal{C}_1 \cup \mathcal{C}_2, t) = \mathcal{FC}(\mathcal{C}_1, t) + \mathcal{FC}(\mathcal{C}_2, t)$.

The additivity property follows from the fact that each cluster can be expressed as a sum of its individual components. The temporal multiplicity property is defined as follows:

*Observation 2.2.* Consider the cluster structure at the time $\mathcal{FC}(\mathcal{C}, t)$. If no points are added to $\mathcal{C}$ in the time interval $(t, t + \delta t)$, them $\mathcal{FC}(\mathcal{C}, t + \delta t) = e^{-\lambda \delta t} \cdot \mathcal{FC}(\mathcal{C}, t)$.

We note that this property holds because of the exponential decay of each component of the cluster structure.

Since the algorithm in this paper is designed for projected clustering of data streams, a set of dimensions is associated with each cluster. Therefore, with each cluster $\mathcal{C}$, we associate a $d$-dimensional bit vector $\mathcal{B}(\mathcal{C})$ which corresponds to the relevant set of dimensions in $\mathcal{C}$. Each element in this $d$-dimensional vector has a 1-0 value corresponding to whether or not a given dimension is included in that cluster. This bit vector is required for the book-keeping needed in the assignment of incoming points to the appropriate cluster. As the algorithm progresses, this bit vector varies in order to reflect the changing set of dimensions. In the next section, we will discuss the clustering algorithm along with the various procedures which are used for cluster maintenance.

## 3. The high dimensional projected clustering algorithm

In this section, we will discuss how the individual clusters are maintained in an online fashion. The algorithm for high-dimensional clustering utilizes an iterative approach which continuously determines new cluster structures while re-defining the set of dimensions included in each cluster.

At the beginning of the clustering process, we run a *normalization process* in order to weigh different dimensions correctly. This is because the clustering algorithm needs to pick the dimensions which are specific to each cluster by comparing the radii along different dimensions. We note that different dimensions may refer to different scales of reference such as age, salary or other attributes which have vastly different ranges and variances. Therefore, it is not possible to compare the dimensions in a meaningful way using the original data. In order to be able to compare different dimensions meaningfully, we perform a normalization process. The aim is to equalize the standard deviation along each dimension. We use an initial sample of the data points to calculate the standard deviation $\sigma_i$ of each dimension $i$. Subsequently, the value of dimension $i$ for each data point is divided by $\sigma_i$. We note that since the data stream may evolve over time, the values of $\sigma_i$ may change as well. Therefore, the normalization factor is recomputed on a periodic basis. Specifically, this process is repeated at an interval of every $N'$ points. However, whenever the value of $\sigma_i$, changes, the corresponding fading cluster statistics may also need to be changed. Let us assume that the standard deviation of dimension $i$ changes from $\sigma_i$ to $\sigma_i'$ during a normalization phase. Then, the cluster statistics $\mathcal{FC}(\mathcal{C}, t) = (\overline{FC2^x(\mathcal{C}, t)}, \overline{FC1^x(\mathcal{C}, t)}, W(t))$ for each cluster $\mathcal{C}$ needs to be correspondingly modified. Specifically, the $i$th entry in $(\overline{FC2^x(\mathcal{C}, t)}$ needs to be multiplied by $\sigma_i^2/\sigma_i'^2$, whereas the $i$th entry in $\overline{FC1^x(\mathcal{C}, t)}$ needs to be multiplied by $\sigma_i/\sigma_i'$.

In figure 1, we have illustrated the basic (incremental) algorithm for clustering high-dimensional data streams. Thus, the incremental pseudo-code shows the steps associated with adding one point to the data stream. The input to the algorithm includes the current

**Algorithm** HPStream (Data Stream Point: $\overline{X}$, Cluster Structures: $\mathcal{FCS}$,
Dimensionality Vector Sets: $\mathcal{BS}$, MaxClusters: $k$, Dimensionality: $l$);
**begin**
{ Assume that $\mathcal{FCS}$ contains the relevant cluster structures denoted by $\mathcal{FCS} = \{\mathcal{FC}^x(\mathcal{C}_1, t) \ldots \mathcal{FC}^x(\mathcal{C}_r, t) \ldots \}$ }
{ Assume that $\mathcal{BS}$ contains the relevant cluster dimensions denoted by $\mathcal{BS} = \{\mathcal{B}(\mathcal{C}_1) \ldots \mathcal{B}(\mathcal{C}_r) \ldots \}$
Receive the next data point $\overline{X}$ at current time $t$ from stream $\mathcal{DS}$;
$\mathcal{BS}$ =*ComputeDimensions*($\mathcal{FCS}, l, \overline{X}$);
**for** $r = 1$ to $|\mathcal{FCS}|$ **do**
$ds(r) = FindProjectedDist(FC^x(\mathcal{C}_r, t), \mathcal{B}(\mathcal{C}_r, \overline{X}))$;
$index = \text{argmax}_i\{ds(i)\}$;
$s = FindLimitingRadius(FC^x(\mathcal{C}_{index}, t, ), \mathcal{B}(\mathcal{C}_{index}))$;
**if** $ds(index) > s$
**then** set $index = |\mathcal{FCS}| + 1$ and add new fading cluster structure $\mathcal{C}_{|\mathcal{FCS}|+1}$ with a solitary data point to $\mathcal{FCS}$;
**else** add $\overline{X}$ to $FC^x(\mathcal{C}_{index}, t)$;
Remove those clusters from $\mathcal{FCS}$ which have zero dimensions assigned to them;
**if** $|\mathcal{FCS}| > k$
**then** delete the least recently added cluster in $\mathcal{FCS}$;
**end;**

*Figure 1.* Basic algorithm for clustering high-dimensional data streams.

cluster structure $\mathcal{FCS}$, and the sets of dimensions associated with each cluster. These cluster structures and sets of dimensions are dynamically updated as the algorithm progresses. The set of dimensions $\mathcal{BS}$ associated with each cluster includes a $d$-dimensional bit vector $\mathcal{B}(\mathcal{C}_i)$ for each cluster structure in $\mathcal{FCS}$. This bit vector contains a 1 bit for each dimension which is included in cluster $\mathcal{C}_i$. In addition, the maximum number of clusters $k$ and the average cluster dimensionality $l$ is used as an input parameter. The average cluster dimensionality $l$ represents the average number of dimensions used in the cluster projection.

The data stream clustering algorithm utilizes an iterative approach by assigning data points to the closest cluster structure at each step of the algorithm. The closest cluster structure is determined by using a *projected distance measure*. For each cluster, only those dimensions which are relevant to that cluster are utilized in the distance computation. At the same time, we continue to re-define the set of projected dimensions associated with each cluster. The re-definition of the projected dimensions aims to keep the radii of the clusters over the projected dimensions as low as possible. Thus, *the clustering process requires a simultaneous maintenance of the clusters as well as the set of dimensions associated with each cluster*.

We will now proceed to systematically describe the steps of the high-dimensional clustering algorithm. A pseudo-code of the algorithm is described in figure 1.

- The set of dimensions associated with each cluster are updated using the procedure *ComputeDimensions*. This procedure determines the dimensions in such a way that the spread along the chosen dimensions is as small as possible. We note that many of the clusters may contain only a few points. This makes it difficult to compute the dimensions in a statistically robust way. In the extreme case, a cluster may contain only one point. In this degenerate case, the computation of the dimensions is not possible since the radii along different dimensions cannot be distinguished. In order to deal with such degenerate cases, we need to use the incoming data point $\bar{X}$ during the determination of the dimensions for each cluster. It is desirable to pick the dimensions in such a way that $\bar{X}$ fits the selected cluster well even after the projected dimensions are selected. Specifically, the data point $\bar{X}$ is temporarily added to each possible cluster during the process of determination of dimensions. This makes significant difference to the chosen dimensions for clusters which contain very few data points. Once these selected dimensions have been chosen, the corresponding bits are stored in $\mathcal{BS}$.
- The next step is the determination of the closest cluster structure to the incoming data point $\bar{X}$. In order to do so, we compute the distance of $\bar{X}$ to each cluster centroid using only the set of projected dimensions for the corresponding cluster. This data in $\mathcal{BS}$ is used as a book-keeping mechanism to determine the set of projected dimensions for each cluster during the distance computation. The corresponding procedure is referred to as *FindProjectedDist*. We will discuss more details about this procedure slightly later.
- Once it is decided which cluster the data point $\bar{X}$ should be assigned to, we determine the natural *limiting radius* of the corresponding cluster. The limiting radius is considered a natural boundary of the cluster. Data points which lie outside this natural boundary are not added to the cluster. Instead such points create new clusters of their own. The procedure for determination of the limiting radius is denoted by *FindLimitingRadius*.

**Algorithm** *FindProjectedDist*($FadedClusterStructure : FC^x(\mathcal{C}_r, t), Bitvector : \mathcal{B}(\mathcal{C}_r), Datapoint : \overline{X}$);
**begin**
{ This procedure finds Manhattan Segmental Distance along the projected dimensions }
**for each** dimension with bit value of 1 in $\mathcal{B}(\mathcal{C}_r)$
find the distance between $\overline{X}$ and the centroid of $\mathcal{B}(\mathcal{C}_r)$;
**return** average distance along the included dimensions;
**end**

*Figure 2*. Finding the projected distance.

- If the incoming data point lies inside the limiting radius, it is added to the cluster. Otherwise, a new cluster needs to be constructed containing the solitary data point $\bar{X}$. We note that if the new data point is noise, the newly created cluster will subsequently have few points added to it. As explained below, this will ultimately lead to the deletion of that cluster.
- In the event that a new cluster is created, the total number of cluster structures in $\mathcal{FCS}$ may increase. Therefore, one cluster needs to be deleted in order to make room for the incoming cluster. In that case, the cluster structure to which the least recent updating was performed is deleted. Thus rule ensures that only stale and outdated clusters are removed by the update process.

In order to determine the closest cluster to the incoming data point, we use the procedure for determining the projected distance of $\bar{X}$ from each cluster $C_r$. The method for finding this distance is discussed in the procedure *FindProjectedDist*, and is illustrated in figure 2. In order to find the projected distance, the distance along each dimension with bit value of 1 in $\mathcal{B}(\mathcal{C}_r)$ is determined. The average distance along these dimensions (also known as the *Manhattan Segmental Distance* (Aggarwal et al., 1999)) is reported as the projected distance. We note that it is not necessary to normalize the distance measurements at this point, since the entire stream has already been normalized at this point. This distance value is computed for each cluster, and the data point $\bar{X}$ is added to the cluster with the least distance value.

In figure 3, we have illustrated the process of computation of the projected dimensions. This is accomplished by calculating the spread along each dimension for each cluster in $\mathcal{FCS}$. This spread is defined as the standard deviation along the corresponding dimension. We note that the fading cluster structure contains the first and second order moments of the data points inside the clusters. The average square radius along the dimension $j$ is given by:

$$r_j^2 = \overline{FC2^x(\mathcal{C}, t)}_j / W(t) - \overline{FC1^x(\mathcal{C}, t)}_j * \overline{FC1^x(\mathcal{C}, t)}_j / W(t)^2. \quad (1)$$

**Algorithm** *ComputeDimensions*(Faded Cluster Structures:$\mathcal{FCS}$, NumberofDimensions: $l$, Incoming Point: $\overline{X}$);
**begin**
Create $|\mathcal{FCS}|$ (tentative) fading cluster structures by adding $\overline{X}$ to each of the existing clusters;
Compute the $|\mathcal{FCS}| * d$ radii of each of the $|\mathcal{FCS}|$ (tentative) clusters along each of the $d$ dimensions;
Pick the $|\mathcal{FCS} * l|$ dimensions with the least radii;
Create a bitvector $\mathcal{B}(\mathcal{C}_r)$ for each cluster $\mathcal{C}_r$ reflecting its projected dimensions;
**end**;

*Figure 3*. Computing the projected dimensions.

**Algorithm** *FindLimitingRadius*(Faded Cluster Structure: $FC^x(\mathcal{C}_{index}, t)$, Bitvector: $\mathcal{B}(\mathcal{C}_{index})$)
**begin**
{ Find the radius $r'$ of the cluster using only the dimensions contained in $\mathcal{B}(\mathcal{C}_{index})$;}
$r_j^2 = \overline{FC2^x(\mathcal{C},t)}_j/W(t) - \overline{FC1^x(\mathcal{C},t)}_j * \overline{FC1^x(\mathcal{C},t)_j}/W(t)^2$;
$R = \sum_{j \in \mathcal{B}(\mathcal{C})} r_j^2$;
Let $d'$ be the number of bits in $\mathcal{B}(\mathcal{C})$ with value of 1;
$R = \sqrt{R/d'}$;
**return**$(R * \tau;)$
**end**

*Figure 4*. Finding the limiting radius of the cluster.

The spread along dimension *j* is defined as the square root of the right hand side of Eq. (1). A total of $|\mathcal{FCS}| * d$ such values are computed and ranked in increasing order. We select the $|\mathcal{FCS}| * l$ dimensions with the least radii as the projected dimensions for that cluster. The incoming data point $\bar{X}$ is included in each cluster for the purpose of computation of dimensions. This ensures that if the incoming data point is added to that cluster, the corresponding set of projected dimensions reflect the included data point $\bar{X}$. This helps in a more stable computation of the projected dimensionality when the cluster contains a small number of data points.

The procedure for finding the limiting radius is illustrated in figure 4. The motivation for finding the limiting radius is to determine the natural boundary of the clusters. Incoming data points which do not lie within this limiting radius of their closest cluster must be assigned a cluster of their own. This is because these data points do not naturally fit inside any of the existing clusters. The limiting radius is defined as a certain factor $\tau$ of the average radius of the data points in the cluster. This radius can be computed using the statistics in the fading cluster structure.

The square radius (as in Eq. (1)) over the dimensions included in $\mathcal{B}(\mathcal{C})$ is averaged in order to find the total square radius of the included dimensions. The square root of this value is the relevant radius of the cluster along the projected set of dimensions. Thus, we find $R = \sqrt{\sum_{J \in \mathcal{B}(\mathcal{C})} r_j^2/d'}$. Here $d'$ is the number of dimensions included in that projected cluster. This value is scaled by a boundary factor $\tau$ in order to decide the final value of the limiting radius. Thus, any incoming data point which lies outside a factor $\tau$ of the average radius along the projected dimensions of its closest cluster needs to create a new cluster containing a solitary data point.

We note that whenever a data point is assigned to a cluster, it needs to be added to the statistics of the corresponding cluster. For this purpose, we need to use the additive and temporal multiplicity properties. The temporal multiplicity is applied in a lazy way at specific instants when a new data point is added to a cluster. Thus, the temporal component of the cluster statistics may remain stale in many cases. However, this does not affect the execution of the overall algorithm. This is because the computation of other measures such as finding the projected distance or computing the dimensions is not affected by the temporal decay factor. The first step in assigning a data point to a cluster is to update the temporal decay function for each cluster. Let $t$ be the current time and $t^{up}$ be the last update time for that cluster. Then, each item in the fading cluster structure is multiplied by the factor $e^{-\lambda \cdot (t - t^{up})}$. At this point, the statistics for the incoming data point are added to

the corresponding fading cluster structure statistics. The additivity property ensures that the updated cluster is represented by these statistics.

At the beginning of the data stream clustering process, it is necessary to perform an additional initialization process by which the original clusters are created. For this purpose, a certain initial portion (containing *InitNumber* points) is utilized. An offline process is used in order to create the initial clusters. This process is implemented as a $K$-means algorithm on an initial sample of the data points. First, a full dimensional $K$-means algorithm is applied to the data points so as to create the initial set of clusters. This full dimensional algorithm is only used as a starting point, which is then subsequently used to find the projected clusters on the initial set of points. Then, the *ComputeDimensions* procedure is applied in order to determine the most relevant dimensions for each cluster. The set of dimensions associated with each cluster is used to compute a new set of assignments of data points to the corresponding centroids. We note that this new assignment is different from the full dimensional assignments, since the set of projected dimensions are used in order to calculate the closest centroid to each data point. These new assignments are utilized to create a new set of $K$ centers. The process of recomputing the dimensions and the centroids is repeated iteratively until the procedure converges to a final set of clusters. This iterative process is necessary in order to improve the quality of the initialization step. These clusters are used to create the fading cluster structures at the beginning of the data stream computation.

We observe that the number of projected dimensions $l$ is used as an input parameter. The *ComputeDimensions* procedure uses this input parameter in picking the $|\mathcal{FCS}| * l$ dimensions with the least radii. Instead of using a fixed number of projected dimensions based on the radius rank, we can use a threshold on the radii of the different dimensions. This would allow the number of projected dimensions to vary over the course of the execution of the data stream clustering process. The use of such a threshold can often be more intuitively appealing over a wide variety of data sets. Since the data normalization ensures that the standard deviation along each dimension is one unit, the threshold can be chosen in terms of the number of standard deviations per dimension. While there may be some variation across data sets in picking this value, this choice has better statistical interpretation.

## 4. Empirical results

In this section we present our thorough experimental study in evaluating the various aspects of HPStream algorithm. All the experiments were performed on an Intel Pentium IV processor computer with 256 MB memory and running on Windows XP professional. In Aggarwal et al. (2003), the CluStream algorithm was proposed, which has shown better clustering quality than the previously designed STREAM clustering algorithm (O'Callaghan et al., 2002). In testing the clustering accuracy and efficiency, we compared our HPStream algorithm with CluStream. We implemented both algorithms using Microsoft Visual C++ 6.0.

In the experiments, the algorithm parameters for CluStream were chosen the same as those adopted in Aggarwal et al. (2003) except that it maintained the same number of *micro-clusters* as that of the *fading cluster structures* used by HPStream. Unless otherwise mentioned, the parameters for HPStream were set as follows: decay-rate $\lambda = 0.5$, spread radius factor $\tau = 2$, $InitNumber = 2000$. Both real and synthetic datasets were

used in evaluating HPStream's clustering quality, stream processing rate, scalability, and sensitivity.

In the experiments, we adopted two popularly used measures to evaluate the clustering quality. The first one is *accuracy*, which is defined by the number of correctly clustered instances (i.e., the instances with dominant class labels in the computed clusters) as a percentage of the total number of data points which arrive within a pre-defined window of time from the current instant. In the following, we also call a predefined window of time a *horizon* and a *horizon* with a size $H$ is composed of $H$ time units. Specifically, let the number of data points in the specified horizon $H$ be $N_H$, the number of clusters in the clustering solution of a clustering algorithm (HPStream or CluStream) be $m$, the number of distinct classes to which the data points in the $i$-th cluster belong be $K_i$, the $j$-th class label in the $i$-th cluster be $l_i^j$, the number of data points in the $i$-th cluster which have a class label $l_i^j$ be $N_{il_i^j}$, the accuracy is defined as follows.

$$\frac{\sum_{i=1}^{m} \max_{j=1}^{k_i} N_{il_i^j}}{N_H}$$

The second measure is *cross entropy*, and a low *cross entropy* indicates a high clustering quality. The *cross entropy* is defined as follows.

$$\sum_{i=1}^{m} \left( \frac{\sum_{q=1}^{k_i} N_{il_i^q}}{N_H} \right) \left( -\frac{1}{\log(k_i)} \sum_{j=1}^{k_i} \left( \frac{N_{il_i^j}}{\sum_{q=1}^{k_i} N_{il_iq}} \right) \log \left( \frac{N_{il_i^j}}{\sum_{q=1}^{k_i} N_{il_i^q}} \right) \right)$$

Because the entropy of a cluster of points characterizes the purity of the corresponding cluster, the cross-entropy measures the average purity over all the clusters, and a low cross entropy indicates a high average purity of the clusters (corresponding to a high clustering quality). The cross-entropy computation normalizes the resulting computations in a statistically more robust way in order to provide a better qualitative overview of the data behavior than raw accuracy computations.

*Real datasets.* In the experiments, we used two real datasets. The first one is the KDD-CUP'99 *Network Intrusion Detection* stream dataset which has been used to evaluate the clustering quality for several stream clustering algorithms (O'Callaghan et al., 2002; Aggarwal et al., 2003). This dataset corresponds to the important problem of automatic and real-time detection of cyber attacks and consists of a series of TCP connection records from two weeks of LAN network traffic managed by MIT Lincoln Labs. Each record can either correspond to a normal connection, or an intrusion which can be classified into one of 22 types. Most of the connections in this dataset are *normal*, but occasionally there could be a burst of attacks at certain times. Also, this dataset contains 494020 connection records, and each connection record has 42 attributes. As in O'Callaghan et al. (2002) and Aggarwal et al. (2003), all 34 continuous attributes will be used for clustering and one outlier point has been removed.

The second real dataset we tested is the *Forest CoverType* dataset and was obtained from the UCI machine learning repository website (i.e., http://www.ics.uci.edu/∼mlearn). This

dataset contains 581012 observations and each observation consists of 54 attributes, including 10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables. In our testing, we used all 10 quantitative variables. There are seven forest cover type classes.

*Synthetic datasets.* We also generated several synthetic datasets to test the clustering quality, efficiency and scalability. Because we know the true cluster distribution *a priori*, we can compare the clusters found with the true clusters and compute the cluster accuracy and cross entropy. The synthetic dataset generator takes four parameters as input: the number of data points $N$, the number of natural clusters $K$, the number of dimensions $d$, and the average number of projected dimensions $l$ (we required $l > \lfloor \frac{d}{2} \rfloor$). The number of projected dimensions in each cluster is uniformly distributed and drawn from $[l - x, l + x]$, where $1 \leq x \leq \lfloor \frac{d}{2} \rfloor$ and $(l - x) \geq 2$. The projected dimensions for each cluster were chosen randomly. The data points of each cluster are normally distributed with the mean for each cluster uniformly chosen from $[0, K)$. The standard deviation was defined as $\sqrt{v}$ for each projected dimension of any cluster, and $y \times \sqrt{v}$ (where $y > 1$) for each of the other dimensions, where $v$ was always randomly chosen from [0.5, 2.5] for any dimension. In our experiments, we set parameters $x$ and $y$ at 2 and 3, respectively.

The data points for different clusters were generated at different times according to a predefined probability distribution. In order to reflect the evolution of the stream data over time, we randomly re-computed the probability of the appearance of a certain cluster periodically. We also assume the projected dimensions will evolve a little over time. In order to capture this kind of evolution, we randomly dropped one of the projected dimensions in one of the clusters and replaced it by a new dimension in a (possibly different) cluster. In addition, we will use the following notations in naming the synthetic datasets: 'B' indicates the base size, i.e., the number of data points in the dataset, whereas 'C', 'D', and 'L' indicate the number of natural clusters, the dimensionality of each point, and the average number of projected dimensions, respectively. For example, *B100kC10D50L30* means the dataset contains in total 100K data points of 50-dimensions, belonging to 10 different clusters, and on average, the number of projected dimensions is 30.

The input parameter $k$ for the HPStream and CluStream algorithm was set to the number of natural clusters in the data set. In the case of the synthetic data sets, this corresponds to the number of input clusters, whereas in the case of the real data sets, this corresponds to the number of classes. Therefore, $k$ was set to 7 and 23 for datasets *Forest CoverType* and *Network Intrusion Detection*, respectively. We note that since the number of clusters was set to the same value for both the HPStream and CluStream algorithms, the results are directly comparable in each case.

### *4.1. Clustering evaluation*

Here we present and analyze our experimental results on clustering quality (accuracy and cross entropy) and the efficiency of the comparing algorithms. We have used the cross entropy in addition to the accuracy, because the former is a more accurate measure.

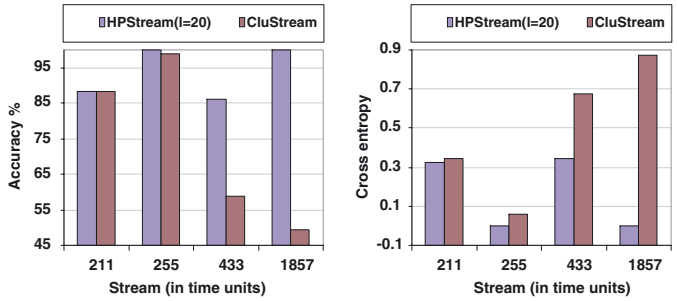



*Figure 5*. Quality comparison (Network Intrusion dataset, horizon = 1, stream_speed = 200).

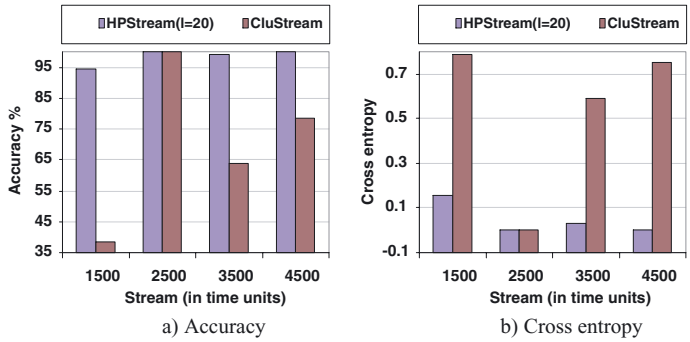



*Figure 6*. Quality comparison (Network Intrusion dataset, horizon = 10, stream_speed = 100).

*Clustering quality.* We evaluated the clustering quality of the HPStream algorithm in comparison with the CluStream algorithm using both real and synthetic datasets.

Figures 5 and 6 show the clustering quality comparison results for the Network Intrusion Detection dataset. In the experiments CluStream used all the 34 dimensions, while we set the average number of projected dimensions at 20 (i.e., $l = 20$) for HPStream, which means on average HPStream used 20 projected dimensions. In figure 5, the stream speed is set at 200 points per time unit and horizon $H = 1$. We chose a series of time points when there were some kind of attack connections happened. For example, at time $T = 211$ there were 1 "*phf*" connection, 23 "*portsweep*" connections, and 176 "*normal*" connections during the past 1 horizon, while at time $T = 1857$, there were totally 79 "*smurf*", 99 "*teardrop*", and 22 "*pod*" attack connections for the last horizon. From Figure 5, we can see that HPStream has a very good clustering quality: its clustering accuracy is always higher than 85% and it has a lower cross entropy than CluStream. For example, at time $T = 1857$, HPStream grouped different attack connections into different clusters, while CluStream grouped all kinds of attacks into one cluster, this is why HPStream has higher accuracy and lower entropy than CluStream. We also set the stream speed at 100 points per time unit and horizon $H$ at 10 to test the clustering quality, figure 6 shows the results. Except at time $T = 2500$, HPStream

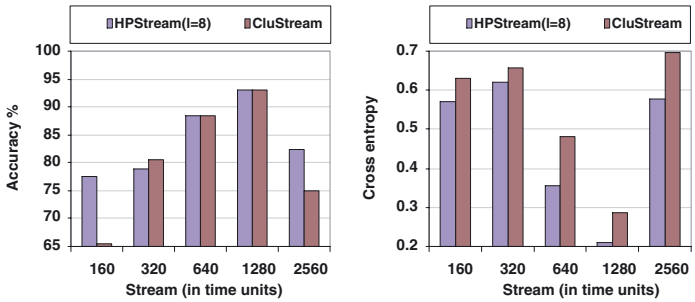



*Figure 7.* Quality comparison (Forest CoverType dataset, horizon = 1, stream_speed = 200).

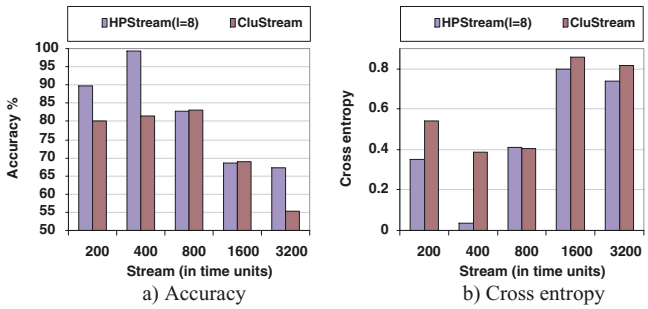



*Figure 8.* Quality comparison (Forest CoverType dataset, horizon = 10, stream_speed = 100).

always has a much higher cluster accuracy and lower entropy than CluStream. We checked the original class labels for the connections in the last ten time units from the current time 2500 and found all the connections belong to one attack type, "*smurf*". As a result, no matter what clustering algorithms we used, they would always have a 100% accuracy.

We also tested the clustering quality of HPStream for another real dataset, *Forest CoverType*. For this dataset, we set the average number of projected dimensions at 8 (i.e., $l = 8$). Figures 7 and 8 show the clustering quality comparison results. In figure 7, we set the stream speed at 200 points per time unit and horizon at 1 (i.e., $H = 1$). Figure 7 shows that although CluStream can achieve similar or even a little better cluster accuracy than HPStream in some cases, its cross entropy is always higher than that of HPStream. We then changed the stream speed to 100 points per time unit and horizon $H$ to 10 and compared the cluster quality for the two algorithms. Figure 8 shows similar results.

We generated one synthetic dataset, *B100kC10D50L30*, to test the clustering quality. This dataset contains 100,000 points that has a total dimensionality of 50 and an average number of projected dimensions 30. The data points belong to 10 different clusters. In the experiments, we set $l$ at 30 for HPStream. As figure 9 shows when we set the stream speed at 200 points per time unit and horizon at 1, HPStream consistently has much better clustering quality than CluStream: HPStream has much higher accuracy and lower cross entropy than

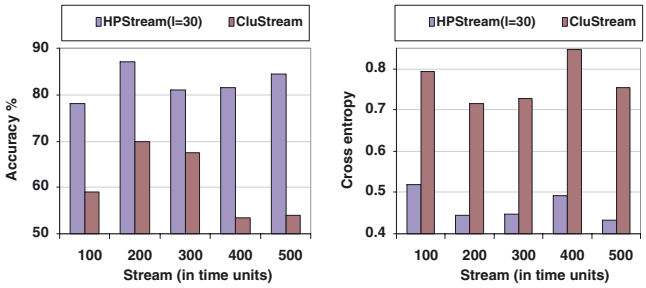



*Figure 9.* Quality comparison (Synthetic dataset B100kC10D50L30, horizon = 1, stream_speed = 200).

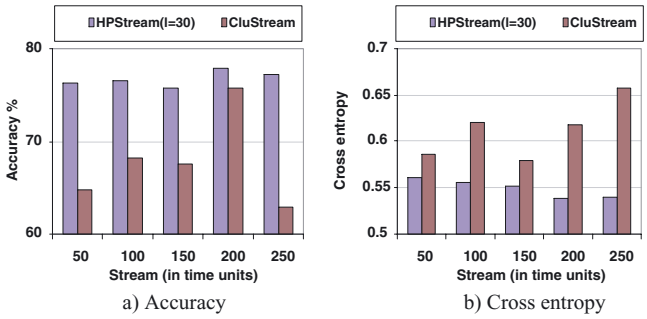



*Figure 10.* Quality comparison (Synthetic dataset B100kC10D50L30, horizon = 10, stream_speed = 400).

CluStream. We then changed the stream speed to 400 points per time unit and used a larger horizon, $H = 10$, to test the clustering quality. Figure 10 shows similar results.

*Clustering efficiency.* We used both the *Network Intrusion Detection* and *Forest CoverType* datasets to test the efficiency of HPStream against CluStream. Because the CluStream algorithm needs to periodically store away the current snapshot of *micro-clusters* under the *Pyramidal Time Framework*, we implemented two versions of the CluStream algorithm: One uses disk to maintain the snapshots of *micro-clusters*, and the other stores the snapshots of *micro-clusters* in memory. The algorithm efficiency is measured by the stream processing rate versus progression of the stream, which is defined as the inverse of the time required to process the last 1000 points (The unit is in points/second). In the experiments, we fixed the stream speed at 200 points per second.

Figure 11 shows the stream processing rate for Network Intrusion dataset, from which we can see that HPStream is more efficient than the disk-based CluStream algorithm and is only marginally slower than the memory-based CluStream algorithm. However, as we know, the memory-based CluStream algorithm will consume much more memory than HPStream. In addition, for this dataset, the processing rate of HPStream is very stable and is around 11,000 points/second, which means HPStream can support a high stream speed at 10,000 points/second. Figure 12 shows the stream processing rate for the *Forest*

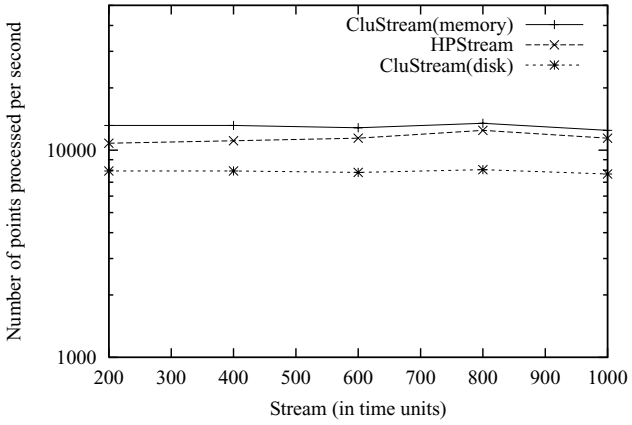



*Figure 11.* Stream Processing Rate (Network Intrusion dataset, stream_speed = 200).

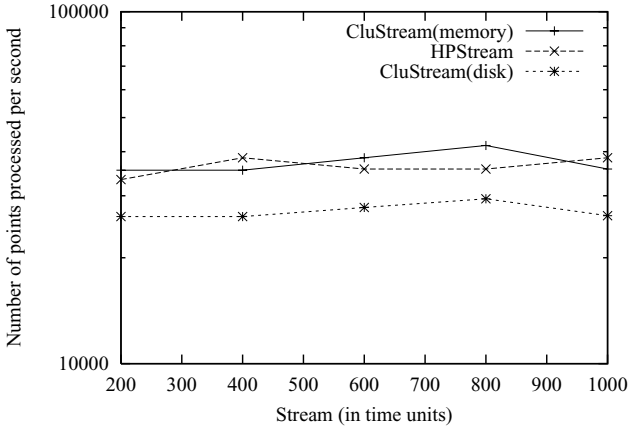
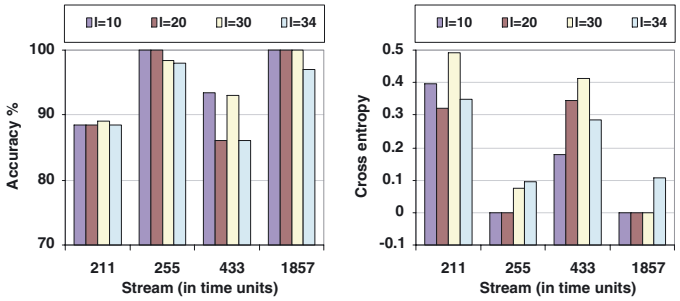



*Figure 12.* Stream Processing Rate (Forest CoverType dataset, stream_speed = 200).

*CoverType* dataset. Because this dataset has a smaller dimensionality than the Network Intrusion dataset, all these algorithms have a higher stream processing rate. For example, both HPStream and the memory-based CluStream algorithms have a stream processing speed around 35,000 points/second. Similarly, HPStream has a higher processing speed than the disk-based CluStream algorithm while consumes less memory than the memory-based CluStream algorithm.

### *4.2. Sensitivity analysis*

In sensitivity analysis, we show how sensitive the clustering quality is in relevance to the average projected dimensionality, the radius threshold, and the decay rate.




a) Accuracy b) Cross entropy

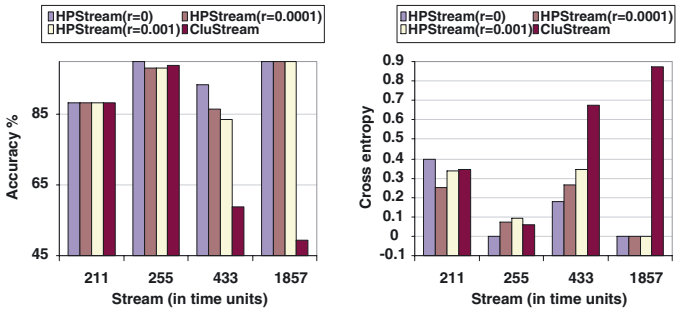
*Figure 13*. Choice of $l$ (Network Intrusion dataset, horizon = 1, stream_speed = 200).

*Choice of the average projected dimensionality l.* The average projected dimensionality $l$ plays an important role in choosing a proper set of projected dimensions that are used by HPStream to do clustering, we want to know how sensitive it is in affecting the clustering quality. Figure 13 shows the results for the Network Intrusion Detection dataset with horizon set at 1 and stream speed set at 200. We can see that the accuracy at a full dimensionality 34 is always (or among) the lowest compared to the projected dimensionality 30, 20, and 10. As to the cross entropy, HPStream with a full dimensionality 34 has the highest entropy at time 255 and 1857 among different projected dimensionality choices, although it works not very bad at time 211 and 433. This shows that as long as we choose a proper projected dimensionality, HPStream can achieve better clustering quality for the real Network Intrusion Detection dataset.

*Choice of the radius threshold.* Although the average projected dimensionality $l$ provides a very flexible and natural way for HPStream to pick the set of well correlated dimensions for clustering high-dimensional data, however, in some cases a radius threshold may be more intuitively chosen as an alternative in selecting the set of projected dimensions. This quality-controlled parameter would allow the number of projected dimensions evolve over the stream. For example, among the 34 dimensions for Network Intrusion Detection dataset, most of them have a deviation 0 for a certain type of connections. If the user has this knowledge in advance, he may choose a radius threshold which is very close to 0 in defining the set of projected dimensions.

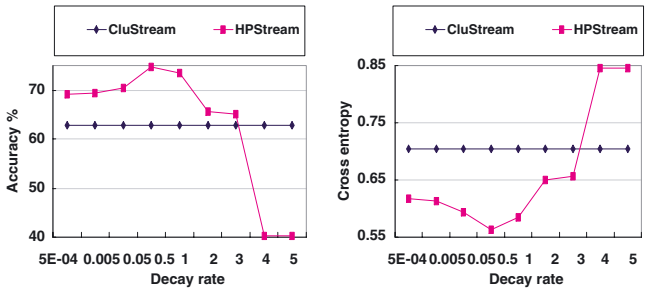
Figure 14 shows the test result for the Network Intrusion dataset by setting the stream speed at 200 points per time unit and horizon $H$ at 1. In the experiments, we test against CluStream the clustering quality of HPStream with varying radius threshold as an input parameter. The result shows that if we set the radius threshold at a small value in the range [0, 0.0001], HPStream has better clustering quality than CluStream: HPStream has an overall higher accuracy and lower entropy than CluStream.

*Choice of the decay rate λ.* Another important parameter for HPStream is the decay rate $\lambda$, which defines the importance of the historical data. In Section 4.1, we set $\lambda$ at a moderate value, 0.5, with which HPStream showed much better clustering quality than CluStream.




a) Accuracy b) Cross entropy

*Figure 14.* Quality comparison based on the radius threshold (Network Intrusion dataset, horizon = 1, stream_speed = 200).

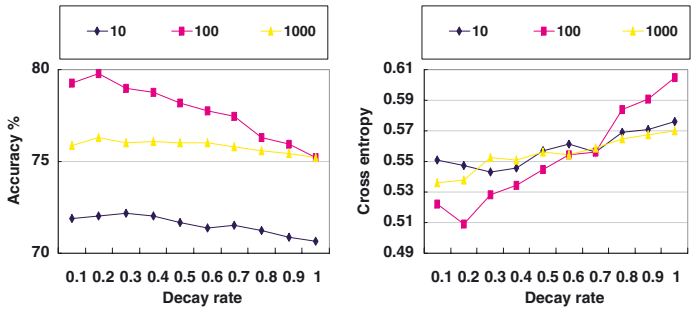



a) Accuracy b) Cross entropy

*Figure 15.* Choice of decay rate λ (Synthetic dataset B100kC10D50L30, stream_speed = 200, H = 10, time units = 100, $l = 30$).

We also did several experiments to isolate the effect of decay rate λ by changing λ from a small value to a large one. We used the synthetic dataset *B100kC10D50L30* and set the stream speed at 200 points per time unit and average projected dimensionality $l = 30$ to test the cluster purity of HPStream at time $T = 100$ with horizon 10. Figure 15 shows the results corresponding to a series of decay rates, 0.0005, 0.005, 0.05, 0.5, 1, 2, 3, 4, and 5. If $0.0005 \leq \lambda \leq 3$, HPStream has a relatively stable cluster quality which is better than that of CluStream. However, when we use a very high value for λ like 4 or 5, HPStream's quality deteriorates quickly, and becomes much worse than that of CluStream.

We also tested the impact of the stream evolution speed on the choice of decay rate λ. In the experiments, we generated three datasets with the same specification as the one used in figure 15 except that they have three different levels of evolution, that is, the probability of the appearance of different clusters was randomly re-computed every 10 points, 100 points and 1000 points, and these datasets are denoted by B100kC10D50L30P10, B100kC10D50L30P100, and B100kC10D50L30P1000, respectively. We computed the accuracy and cross entropy for HPStream at time $T = 100$ by setting the stream_speed at 200 points per time unit and horizon at 10, and varying the decay rate λ from 0.1 to 1.




a) Accuracy b) Cross entropy

*Figure 16.* Impact of the evolution speed on the choice of decay rate $\lambda$ (Synthetic datasets B100kC10D50L30Px, stream_speed = 200, H = 10, time units = 100, $l = 30$).

Figure 16 shows the results. We can see that for different datasets we need to adjust $\lambda$ in order to achieve the optimal performance. For example, for dataset B100kC10D50L30P10, $\lambda = 0.3$ can lead to the highest accuracy and the lowest cross entropy, while for dataset B100kC10D50L30P100, the highest accuracy and the lowest entropy can be achieved with $\lambda = 0.2$.

### *4.3. Scalability test*

The scalability tests presented below show that HPStream is linearly scalable with both dimensionality and the number of clusters.

Figures 11 and 12 have shown that HPStream has very stable stream processing speed along with the progression of the stream for the two real datasets, which means HPStream has very good scalability in terms of base size. High scalability in terms of dimensionality and the number of clusters is also very critical to the success of a high-dimensional clustering algorithm. We generated a series of synthetic datasets to test the scalability of HPStream.

We first generated 2 series of datasets with varying number of dimensions to test the scalability against dimensionality. These data sets are denoted by B200kC10 and B400kC20 respectively. For each series of datasets, we generated 4 datasets with dimensionality $d$ set at 10, 20, 40, and 80, respectively. The average number of projected dimensions for each dataset is set at $0.8 \times d$ and the stream speed is set at 100 points per time unit. Figure 17 shows that when we varied the dimensionality from 10 to 80, both HPStream and CluStream (this is the disk-based implementation) have linear increase in runtime for datasets with different number of points and different number of clusters. For example, for dataset series B200kC10, the runtime for HPStream increases from 6.579 seconds to 49.401 seconds when the dimensionality is changed from 10 to 80. This difference is because of the extra effort of maintaining the pyramidal time frame by the CluStream algorithm.

To test the scalability against the number of clusters, we generated another 2 series of datasets with varying number of clusters, B200kD20 and B400kD40. For each series of datasets, we generated 4 datasets with the number of natural clusters set at 5, 10, 20,

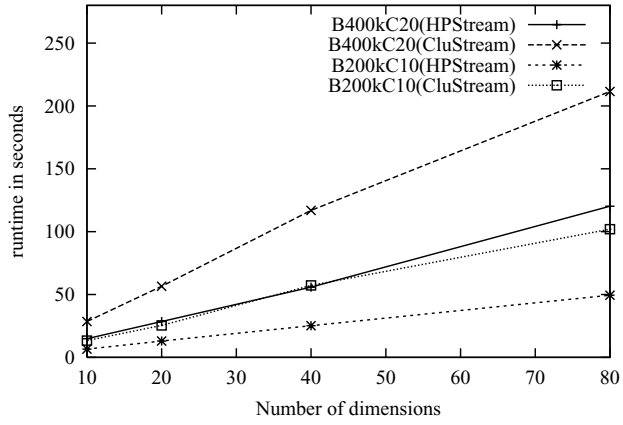



*Figure 17.* Scalability with dimensionality (stream_speed $= 100, l = 0.8 \times d$).

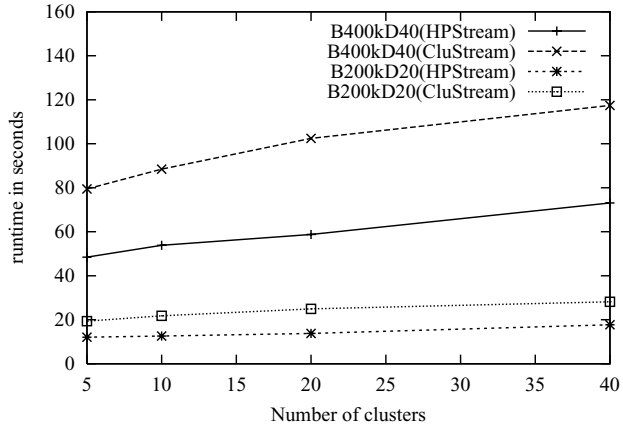



*Figure 18.* Scalability with number of clusters (stream_speed $= 100, l = 0.6 \times d$).

and 40, respectively. The value of the input clusters was set to the number of natural clusters. The average number of projected dimensions for each dataset is set at $0.6 \times d$ and the stream speed at 100 points per time unit. Figure 18 shows that the runtime of both HPStream and the disk-based CluStream algorithms has very good scalability in terms of the number of clusters for datasets with different number of points and dimensionality. The high scalability of HPStream in terms of the number of clusters stems from both the algorithm design and implementation. Among the three most costly functions in HPStream algorithm, the computation of *FindLimitingRadius* has nothing to do with the number of clusters, *FindProjectedDist* is linearly scalable to the number of clusters, whereas for *ComputeDimensions*, we can exploit the temporal locality to improve its efficiency. At a certain period, the points usually only belong to a small number of clusters, and only the

dimensions of these clusters will be changed during the past period with the necessity to re-compute their radii.

## 5. Discussion

Our experiments have shown that the HPStream framework leads to accurate and efficient high-dimensional stream clustering. This framework can be extended in many ways to assist stream data mining.

First, some methodologies, such as the cluster structure and micro-clustering ideas, though designed for projected stream clustering, can be applied to projected clustering of non-stream data as well. Moreover, the method worked out here for high-dimensional projected stream clustering represents a general methodology, independent of particular evaluation measures and implementation techniques. For example, one can change the distance measure from Euclidean distance to other measures, or change detailed clustering algorithm, such as *k*-means, to other methods, the general methodology should still be applicable. However, it is interesting to work out the detail implementation techniques for particular applications.

Second, one extension of the framework is to use tilted time windows to store data at different time granularity. This may take somewhat more space in cluster structure, however, it may give user more flexibility to dynamically assign or modify fading ratio, as well as to discover clusters at more flexibly specified windows or time periods to facilitate the discovery of cluster evolution regularity.

Finally, this study may promote the development of new streaming data mining functions, such as stream classification and similarity analysis based on dynamically discovered projected clusters.

## 6. Conclusions

We have presented a new framework, HPStream, for high-dimensional projected clustering of data streams. It finds projected clusters in particular subsets of the dimensions by maintaining condensed representations of the clusters over time. The algorithm provides better quality clusters than full dimensional data stream clustering algorithms. We tested the algorithm on a number of real and synthetic data sets. In each case, we found that the HPStream algorithm was more effective than the full dimensional CluStream algorithm.

High-dimensional projected clustering of data streams opens a new direction for exploration of stream data mining. With this methodology, one can treat projected clustering as a preprocessing step, which may promote more effective methods for stream classification, similarity, evolution and outlier analysis.

## References

Aggarwal, C.C. 2004. A human-computer interactive method for projected clustering. IEEE Transactions on Knowledge and Data Engineering, 16(4):448–460.

Aggarwal, C.C., Procopiuc, C., Wolf, J., Yu, P.S., and Park, J.-S. 1999. Fast algorithms for projected clustering. In ACM SIGMOD Conference.

Aggarwal, C.C., Han, J., Wang, J., and Yu, P. 2003. A framework for clustering evolving data streams. In VLDB Conference.

Aggarwal, C.C. 2002. An intuitive framework for understanding changes in evolving data streams. ICDE Conference.

Aggarwal, C.C. 2003. A framework for diagnosing changes in evolving data streams. ACM SIGMOD Conference, pp. 575–586.

Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. 1998. Automatic subspace clustering of high dimensional data for data mining applications. ACM SIGMOD Conference.

Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. 2002. Models and issues in data stream systems. ACM PODS Conference.

Domingos, P. and Hulten, G. 2000. Mining high-speed data streams. In ACM SIGKDD Conference.

Farnstrom, F., Lewis, J., and Elkan, C. 2000. Scalability for Clustering Algorithms Revisited. SIGKDD Explorations, 2(1):51–57.

Feigenbaum, J., et al. 2000. Testing and spot-checking of data streams. In ACM SODA Conference.

Guha, S., Mishra, N., Motwani, R., and O'Callaghan, L. 2000. Clustering data streams. In IEEE FOCS Conference.

Guha, S., Rastogi, R., and Shim, K. 1998. CURE: An efficient clustering algorithm for large databases. ACM SIGMOD Conference.

Jain, A. and Dubes, R. 1998. Algorithms for clustering data. New Jersey: Prentice Hall.

Ng, R. and Han, J. 1994. Efficient and effective clustering methods for spatial data mining. In Very Large Data Bases Conference.

O'Callaghan, L., Mishra, N., Meyerson, A., Guha, S., and Motwani, R. 2002. Streaming-data algorithms for high-quality clustering. In ICDE Conference.

Zhang, T., Ramakrishnan, R., and Livny, M. 1996. BIRCH: An efficient data clustering method for very large databases. ACM SIGMOD Conference.

**Charu C. Aggarwal** received his B.Tech. degree in Computer Science from the Indian Institute of Technology (1993) and his Ph.D. degree in Operations Research from the Massachusetts Institute of Technology (1996). He has been a Research Staff Member at the IBM T. J. Watson Research Center since June 1996. He has applied for or been granted over 50 US patents, and has published over 75 papers in numerous international conferences and journals. He has twice been designated Master Inventor at IBM Research in 2000 and 2003 for the commercial value of his patents. His contributions to the Epispire project on real time attack detection were awarded the IBM Corporate Award for Environmental Excellence in 2003. He has been a program chair of the DMKD 2003, chair for all workshops organized in conjunction with ACM KDD 2003, and is also an associate editor of the IEEE Transactions on Knowledge and Data Engineering Journal. His current research interests include algorithms, data mining, privacy, and information retrieval.

**Jiawei Han** is a Professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He has been working on research into data mining, data warehousing, stream and RFID data mining, spatiotemporal and multimedia data mining, biological data mining, social network analysis, text and Web mining, and software bug mining, with over 300 conference and journal publications. He has chaired or served in many program committees of international conferences and workshops, including ACM SIGKDD Conferences (2001 best paper award chair, 1996 PC co-chair), SIAM-Data Mining Conferences (2001 and 2002 PC co-chair), ACM SIGMOD Conferences (2000 exhibit program chair), International Conferences on Data Engineering (2004 and 2002 PC vice-chair), and International Conferences on Data Mining (2005 PC co-chair). He also served or is serving on the editorial boards for Data Mining and Knowledge Discovery, IEEE Transactions on Knowledge and Data Engineering, Journal of Computer Science and Technology, and Journal of Intelligent Information Systems. He is currently serving on the Board of Directors for the Executive Committee of ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD). Jiawei has received three IBM Faculty Awards, the Outstanding Contribution Award at the 2002 International Conference on Data Mining, ACM Service Award (1999)

and ACM SIGKDD Innovation Award (2004). He is an ACM Fellow (since 2003). He is the first author of the textbook "Data Mining: Concepts and Techniques" (Morgan Kaufmann, 2001).

**Jianyong Wang** received the Ph.D. degree in computer science in 1999 from the Institute of Computing Technology, the Chinese Academy of Sciences. Since then, he ever worked as an assistant professor in the Department of Computer Science and Technology, Peking (Beijing) University in the areas of distributed systems and Web search engines (May 1999–May 2001), and visited the School of Computing Science at Simon Fraser University (June 2001–December 2001), the Department of Computer Science at the University of Illinois at Urbana-Champaign (December 2001–July 2003), and the Digital Technology Center and Department of Computer Science and Engineering at the University of Minnesota (July 2003–November 2004), mainly working in the area of data mining. He is currently an associate professor in the Department of Computer Science and Technology, Tsinghua University, Beijing, China.

**Philip S. Yu** is the manager of the Software Tools and Techniques group at the IBM Thomas J. Watson Research Center. The current focuses of the project include the development of advanced algorithms and optimization techniques for data mining, anomaly detection and personalization, and the enabling of Web technologies to facilitate E-commerce and pervasive computing. Dr. Yu,s research interests include data mining, Internet applications and technologies, database systems, multimedia systems, parallel and distributed processing, disk arrays, computer architecture, performance modeling and workload analysis. Dr. Yu has published more than 340 papers in refereed journals and conferences. He holds or has applied for more than 200 US patents. Dr. Yu is an IBM Master Inventor. Dr. Yu is a Fellow of the ACM and a Fellow of the IEEE. He will become the Editor-in-Chief of IEEE Transactions on Knowledge and Data Engineering on Jan. 2001. He is an associate editor of ACM Transactions of the Internet Technology and also Knowledge and Information Systems Journal. He is a member of the IEEE Data Engineering steering committee. He also serves on the steering committee of IEEE Intl. Conference on Data Mining. He received an IEEE Region 1 Award for "promoting and perpetuating numerous new electrical engineering concepts". Philip S. Yu received the B.S. Degree in E.E. from National Taiwan University, Taipei, Taiwan, the M.S. and Ph.D. degrees in E.E. from Stanford University, and the M.B.A. degree from New York University.