Incremental Clustering of Dynamic Data Streams Using Connectivity Based Representative Points

Sebastian Lühr * Mihai Lazarescu

Department of Computing, Curtin University of Technology, Kent Street, Bentley 6102, Western Australia

Abstract

We present an incremental graph-based clustering algorithm whose design was motivated by a need to extract and retain meaningful information from data streams produced by applications such as large scale surveillance, network packet inspection and financial transaction monitoring. To this end, the method we propose utilises representative points to both incrementally cluster new data and to selectively retain important cluster information within a knowledge repository. The repository can then be subsequently used to assist in the processing of new data, the archival of critical features for off-line analysis, and in the identification of recurrent patterns.

Key words: Data mining, incremental graph-based clustering, stream data clustering, recurrent change, knowledge acquisition

1 Introduction

Stream data clustering is a challenging area of research that attempts to extract useful information from continuously arriving data [22,12]. Limitations imposed by available computational resources typically constrain our ability to process such data and restrict, if not completely remove, our ability to revisit previously seen data points. Given these conditions we still desire effective means by which to identify trends and changes in evolving stream data. Furthermore, we would like to have the ability to access historical data without requiring any significant storage/processing resources or multiple passes over

* Corresponding author.
Email address: M.Lazarescu@curtin.edu.au (Mihai Lazarescu).

the data, especially since historical data is critical for large scale stream mining applications such as surveillance, network packet classification, financial databases, e-mail analysis and multimedia. Stream data clustering has been heavily investigated in the past years and, although the architectural aspects of processing data streams has received attention [19,20,17,10,56], most effort concentrated on the mining and clustering aspect of the problem [26]. Several algorithms have been developed to handle stream data containing clusters of different shapes, sizes and densities but only a small number can handle difficult clustering tasks without supervision.

Generally, the algorithms require expert assistance in the form of the number of partitions expected or the expected density of clusters. Furthermore, none of these algorithms attempt to build a selective history to track the underlying changes in the clusters observed and, thus, have the disadvantage that they are required to re-learn any recurrently occurring patterns.

The aims of our work have been to develop an algorithm that can handle all types of clusters with minimal expert help and to provide a graph-based description of changes and patterns observed in the stream in order to enable a detailed analysis of the knowledge acquired. The contribution of our work is a highly accurate stream processing algorithm that can handle all types of numeric data while requiring only limited resources. The novelty of our work comes in the form of an integrated knowledge repository which is used in tandem with the stream data clustering algorithm.

In this paper, we present RepStream, a sparse-graph-based stream clustering approach that employs representative cluster points to incrementally process incoming data. The graph-based description is used because it allows us to model the spatio-temporal relationships in a data stream more accurately than is possible via summary statistics. Each cluster is defined by using two types of representative points: exemplar points that are used to capture the stable properties of the cluster and predictor points which are used to capture the evolving properties of the cluster.

A critical aspect of our research has been to avoid having to "re-discover" previously learned patterns by maximising the reuse of previously useful cluster information. For this reason, a repository of knowledge is used to capture the history of the relevant changes occurring in the clusters over time. The use of the repository offers two major benefits.

First, the algorithm can handle recurrent changes in the clusters more effectively by storing a concise representation of persistent and consistent cluster features. These features assist in the classification of new data points belonging to historical cluster distributions within an evolving data stream. The retention of such features is important as they permit the algorithm to discard

older data points in order to adhere to constraints in available memory and computational resources while continuing to store useful cluster features. It is posited that this retention enables the algorithm to handle recurrent change with higher classification accuracy than would be possible if only the most recent cluster trends were retained.

Second, the repository provides a concise knowledge collection that can be used to rebuild a cluster's overall shape and data distribution history. It is therefore possible to archive core cluster features for possible future off-line analysis when a recall of historical changes is desired.

The clustering effectiveness of the algorithm has been extensively investigated and is demonstrated via experimental results obtained clustering both synthetic and dynamic real world benchmark data sets. Furthermore, the clustering quality of the algorithm is shown to produce substantially better results when compared with three of the most popular current stream clustering algorithms: DenStream, HPStream and CluStream.

A preliminary version of this paper appears as [43]. In this paper we present a more detailed explanation of the workings of the algorithm and its relationship to existing works. Furthermore, we offer an extended set of experimental results and investigate the algorithm's parameter sensitivity.

2 Related Work

Several important stream clustering algorithms have been introduced in recent years. One of the first data stream clustering methods to consider the archival of cluster information was CluStream [4]. The algorithm uses microclusters to capture and record statistical summary information suitable for off-line analysis. The amount of information that is archived is controlled by a user specified maximum number of microclusters with the algorithm attempting to capture as much detail as memory constraints allow. CluStream is, however, best suited to situations in which clusters are spherical, reducing the algorithm's suitability for many real world data sets. CluStream has recently been enhanced to deal with uncertainty in the data stream, increasing the accuracy of the algorithm when clustering noisy data sets [7].

HPStream, a modification of CluStream to enable clustering of high-dimensional data, was proposed in [5]. The algorithm employs a data projection method to reduce the dimensionality of the data stream to a subset of dimensions that minimise the radius of cluster groupings. It was demonstrated that by projecting data onto a smaller set of dimensions both synthetic and real world data sets could be more accurately processed. As with CluStream, however,

the underlying assumption remains that clusters in the projected space remain spherical in nature. How best to classify incoming data using the CluStream and HPStream frameworks was discussed in [6].

Birch [54] is a well known hierarchical clustering algorithm that incrementally updates summary cluster information for off-line analysis. Clusters suitable for classification are then extracted using the summary information via a second pass over the data. The algorithm was later adapted for online clustering and classification by combining the secondary off-line phase with the incremental update component [27].

A k-median approximation algorithm designed for stream clustering was introduced in [48,29] while a k-means approach to cluster parallel data streams was presented in[15]. Such algorithms, however, inherit the features typical of medoid based approaches, most notably the requirement that users supply the number of clusters to be searched for and the assumption that cluster shapes are likely to be spherical. Their advantage, however, lies in their low computational complexity and their adaptability to the distributed data clustering domain; see [13] for an example and the references contained therein.

A statistical approach to distributed data stream clustering based on expectation maximisation (EM) was recently introduced in [55]. Here, Gaussian mixture models were used to capture the distribution of data arriving at each host in a peer-to-peer network. The proposed algorithm, CluDistream, employed a test-and-cluster approach to efficiently cluster points, a strategy which requires the mixture models to only be updated on a sample of points that probabilistically belong to an existing cluster. Cluster splits and merges in CluDistream are event driven and managed by a co-ordinator host which receives notifications of significant changes to the processing hosts mixture models. Such notifications allow the algorithm to collect events suitable for later analysis but does not provide the capacity for efficiently coping with recurrent patterns.

The work of [28] introduced DUCstream, a grid-based technique that seeks dense "units", or regions, in a multidimensional space. Clusters are discovered by applying the CLIQUE [8] algorithm to regions that are considered to be dense. The algorithm adapts to changes in the data stream by disregarding regions whose density fades over times. This approach, however, relies on users specifying the granularity of the data processing as the dense unit detection requires data to be processed in blocks.

A more recent grid-based approach was introduced with the Cell Tree in [49]. Each cell in the Cell Tree stores a statistical summary of a region of the data space which is recursively partitioned along a single dimension until a sparsity threshold has been reached. Each cell stores the time-weighted mean and standard deviation of the data distribution within the cell, enabling the

partitioning of each region to cease when the data within the partition is able to be represented. The algorithm is able to adapt to the changing data distributions by further partitioning regions when their density grows or by merging adjacent cells as the data they represent decay.

The literature comprises a considerable number of graph-based clustering approaches [32,36,37,53]. Recent works related to our proposed approach are referred to here.

A well known algorithm, Chameleon [41] uses the hMETIS [40] multilevel graph partitioning algorithm to recursively divide a sparse graph into microclusters. These clusters are then iteratively merged based on user specified thresholds for measures of relative interconnectivity and closeness. A similar approach is used in MOSAIC [21] where representative based clustering such as k-means [44] is used to first find a large collection of smaller clusters which can then be selectively merged via a proximity map to form more complex cluster structures.

More recently, a multi-density clustering technique that extends the DBSCAN [24] density-based clustering approach to stream clustering was proposed in [18]. The algorithm, DenStream, extends DBSCAN by adapting the original density based connectivity search to a microcluster approach. The microclusters allow DenStream to efficiently summarise the overall shape of clusters as the data evolves without requiring the complete set of points to be retained in memory. The final clustering can be obtained for any time step by applying a variant of the DBSCAN algorithm on the microclusters themselves.

An incremental version of DBSCAN was earlier proposed in [23]. As with DBSCAN, the algorithm obtains groupings based on the nearest neighbourhood connectivity of points within an a priori defined radius known as the ϵ-neighbourhood. Clusters are formed by identifying *density-reachable* chainings between sets of well connected points. Incremental DBSCAN is able to cope with arbitrary point insertions and deletions, enabling the algorithm to forget past data. No measure of usefulness of the data is retained, however, limiting the algorithm to keeping only the most recent data points in memory. The algorithm is therefore likely to discard possibly reusable cluster information without consideration towards its value.

The ordering points to identify the cluster structure (OPTICS) algorithm [9], a multi-density extension of DBSCAN, identifies clusters by first ordering points based on their density and connectivity within their ϵ-neighbourhood. Plotting the ordering of the points against a reachability measure allows cluster boundaries to then be visually or algorithmically identified by locating regions of steep change in the reachability plots. OPTICS was recently extended to enable data stream processing with the introduction of OpticsStream [51].

Here, orderings of microclusters are used to adapt the clustering to evolving data distributions and for temporal analysis of the cluster structure. Unfortunately, no unsupervised method for cluster extraction is provided, the method instead requiring manual inspection of generated two- and three-dimensional plots.

Nasraoui et al. [47,46] proposed a clustering technique based on an artificial immune system in which incoming data points are presented to a network of artificial white blood cells (b-cells) that mimic the adaptive mutation observed in biological immune systems. The algorithm classifies incoming data points based on the similarity of each point to existing b-cells and each cell's radius of influence. Cells rapidly evolve with the data stream via an update method that simulates cell stimulation. New b-cells are created whenever no existing b-cell is stimulated by a new data point; the network of b-cells is periodically reduced by means of k-means clustering.

Two alternative measures to the often used minimum cut metric, graph expansion and conductance formed the basis for a graph partitioning scheme in [38,39]. Here graph expansion incorporates the relative size of sub-graphs into the cut measure. Conductance, in contrast, gauges the importance of vertices by assigning them weightings based on their similarity to neighbouring vertices. Clustering attempts to find groupings that result in a minimum threshold for expansion or conductance while simultaneously reducing the sum of the intercluster edge weights in the clustering system. This approach was expanded in [25] with a technique that combines the bicriteria used in [38,39] into a single parameter that bounds both the minimum graph expansion and the maximum intercluster edge weight.

Another technique that has been explored is the analysis of a stochastic process performing a random walk through a weighted graph in order to probabilistically isolate clusters. One such method is the algorithm in [31] which deterministically analyses a similarity matrix to locate separations between vertices on a weighted graph. The Markov clustering (MCL) algorithm described in [52] similarly discovers cluster formations by updating a state transition matrix to simulate random walks. The first random walk method to be proposed, in contrast, required significant space and computational resources to generate actual walks [30].

None of the algorithms mentioned provide a means to selectively archive historical information. Those algorithms that facilitate archiving instead tend to approach the issue by storing summary statistics with which general changes in clusters can be revisited.

3 Preliminaries

Given a data stream P of time ordered points $P = \{p_1, \dots p_{|P|}\}$, we wish to find groupings of points sharing similar properties. We define a cluster c to be a set of points $c = \{p_1 \dots p_{|c|}\}$ where each point p_i is a multidimensional vector $p_i = \{p_{i,1} \dots p_{i,D}\}$ of D dimensions. Let C be the set of clusters $C = \{c_1 \dots c_{|C|}\}$.

Let the set $G = \{g_1 \dots g_{|P|}\}$ consist of the ideal cluster assignments for points P such that the j^{th} element g_j correctly labels point p_j. We aim to assign labels to data points such that each point is correctly classified or any misclassification is minimised. For example, if correct classification of a domesticated cat (*Felis sylvestris*) in a stream of observed animal is not achieved then we would prefer a misclassification such as sand cat (*Felis margarita*) over leopard (*Prionailurus bengalensis*), a member of a different genus. The distance between point p_i and point p_j is given as $\mathrm{D}(p_i, p_j)$ and is typically the Euclidean or Manhattan distance between two points, although other domain specific functions may be desirable [35,1,2].

Points are inserted into a directed k-nearest neighbour (k-NN) sparse graph $\mathrm{SG}(V, E)$ of vertices $V = \{v_1, \dots v_{|V|}\}$ and edges $E = \{e_1, \dots e_{|E|}\}$ such that the i^{th} vertex v_i corresponds to point $p_i \in P$. Each edge is an ordered pair $\langle v, u \rangle$ of vertices such that $v, u \in V$ that represents the distance to a nearest neighbour in the selected dimensions. The sparse graph representation is used as it provides a rich representation of relationships that is otherwise not available by only labelling data points. For example, a surveillance application benefits from knowing the spatio-temporal relationships in a suspect's position over time as opposed to only labels and positions of locations.

Definition 1 *(nearest neighbour) Let NN(v_i) be the set nearest neighbours of a vertex v_i such that*

$$\forall v_x \in V \text{ and } \forall v_j \in NN(v_i), D(v_i, v_j) \leq D(v_i, v_x) \qquad v_x \neq v_j. \tag{1}$$

Let $NN(v_i, j)$ be a function that gives the j^{th} nearest neighbour of v_i.

Definition 2 *(incoming edges) Given an ordered pair $\langle v_j, v_i \rangle$ let IE(v_i) be the set of incoming edges directed at vertex v_i such that*

$$\forall j, \exists \langle v_j, v_i \rangle \in IE(v_i) \text{ and } \exists \langle v_j, v_i \rangle \in E. \tag{2}$$

Let $|IE(v_i)|$ be the number of incoming edges of v_i.

Definition 3 *(outgoing edges) Given an ordered pair $\langle v_i, v_j \rangle$ let OE(v_i) be the*

set of outgoing edges from vertex v_i such that

$$\forall j, \exists \langle v_i, v_j \rangle \in OE(v_i) \text{ and } \exists \langle v_i, v_j \rangle \in E. \tag{3}$$

Definition 4 *(reciprocally connected) Let $RC(v_i)$ be the set of vertices that are reciprocally connected to a vertex v_i such that*

$$\forall v_x \in RC(v_i), \exists \langle v_i, v_x \rangle \in E \text{ and } \exists \langle v_x, v_i \rangle \in E. \tag{4}$$

Figure 2 shows the nearest neighbours and the incoming edges of two sampled vertices from the sparse graph in Figure 1. Reciprocally connected vertices along with outgoing edges of a sample vertex are demonstrated in Figure 3.

Let $R = \{r_1, \dots r_{|R|}\}$ be a set of representative vertices on SG such that $\forall x, r_x \in V$ and let $\mathrm{RSG}(W, F)$ be a directed k-nearest neighbour representative sparse graph which links the vertices $W = \{w_1, \dots w_{|W|}\}$ via edges $F = \{f_1, \dots f_{|F|}\}$. An edge in F is an ordered pair $\langle v, u \rangle$ of vertices such that $v, u \in R$. Let $\mathrm{NN_R}(r_i)$, $\mathrm{NN_R}(r_i, j)$ and $\mathrm{RC_R}(r_i)$ be functions that provide the nearest neighbours, the nearest j^{th} neighbour and the set of vertices that are reciprocally linked to a representative vertex r_i on RSG.

Figure 4 depicts the representative sparse graph RSG formed from the sparse graph SG in Figure 1.

Definition 5 *(predictor) Let a representative r_i be a predictor if r_i satisfies the condition that $|IE(r_i)| < \frac{k}{2}$.*

Definition 6 *(exemplar) Let a representative r_i be an exemplar if r_i satisfies the condition that $|IE(r_i)| \geq \frac{k}{2}$.*

Definition 7 *(representative vertex) Representative vertices represent at most k non-representative vertices on the sparse graph SG. A vertex v_i is made representative if at any time $\nexists j, v_j \in RC(v_i), v_j \in R$, that is, if it is not reciprocally connected to an existing representative. Representatives are further categorised into a set of exemplar representatives $R^E = \{r_1^E, \dots r_{|R^E|}^P\}$ and predictor representatives $R^P = \{r_1^P, \dots r_{|R^P|}^P\}$ such that $R^P \cup R^E = R$ and $R^P \cap R^E = \emptyset$.*

Clustering decisions in RepStream are made via vertices representative of regions within the cluster space. Representative vertex labels are included purely to aid in their interpretation and do not affect the operation of the algorithm. Exemplars are said to represent cluster regions for which we have evidence of both consistency and persistence. Predictors, in contrast, represent regions which possess a potential to become prominent cluster features but where such evidence has not yet been seen.

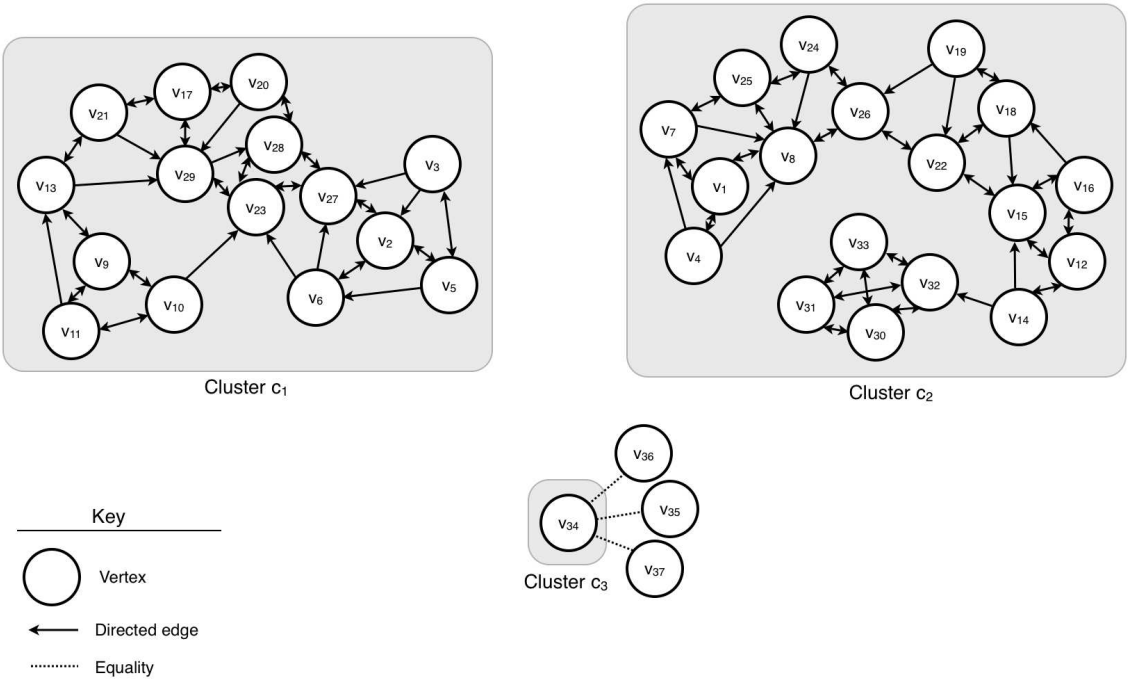


Fig. 1. An example sparse graph SG showing a set of vertices and their directed edges for $k = 3$. Cluster c_3 was formed from a set of identical vertices.

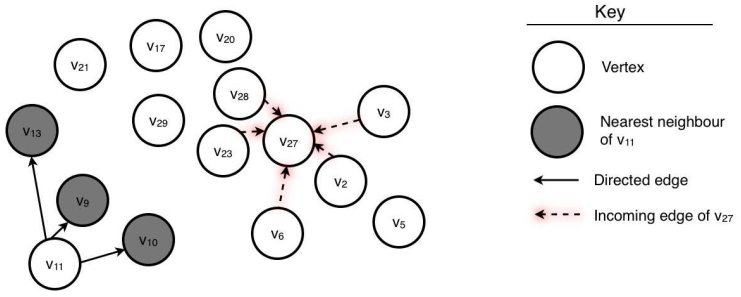


Fig. 2. The nearest neighbours of vertex v_{11} and the incoming edges of vertex v_{27} in cluster c_1 on sparse graph SG for $k = 3$.

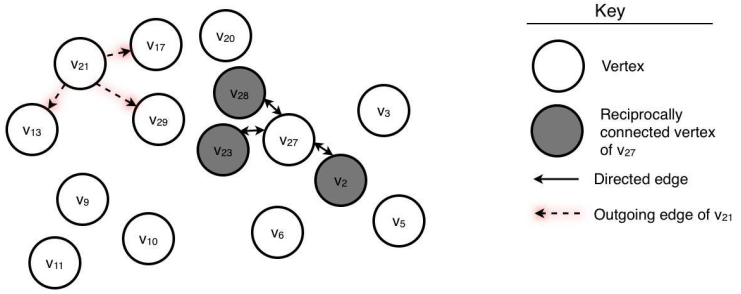


Fig. 3. The vertices reciprocally connected to vertex v_{27} and the outgoing edges of vertex v_{21} in cluster c_1 on sparse graph SG for $k = 3$.

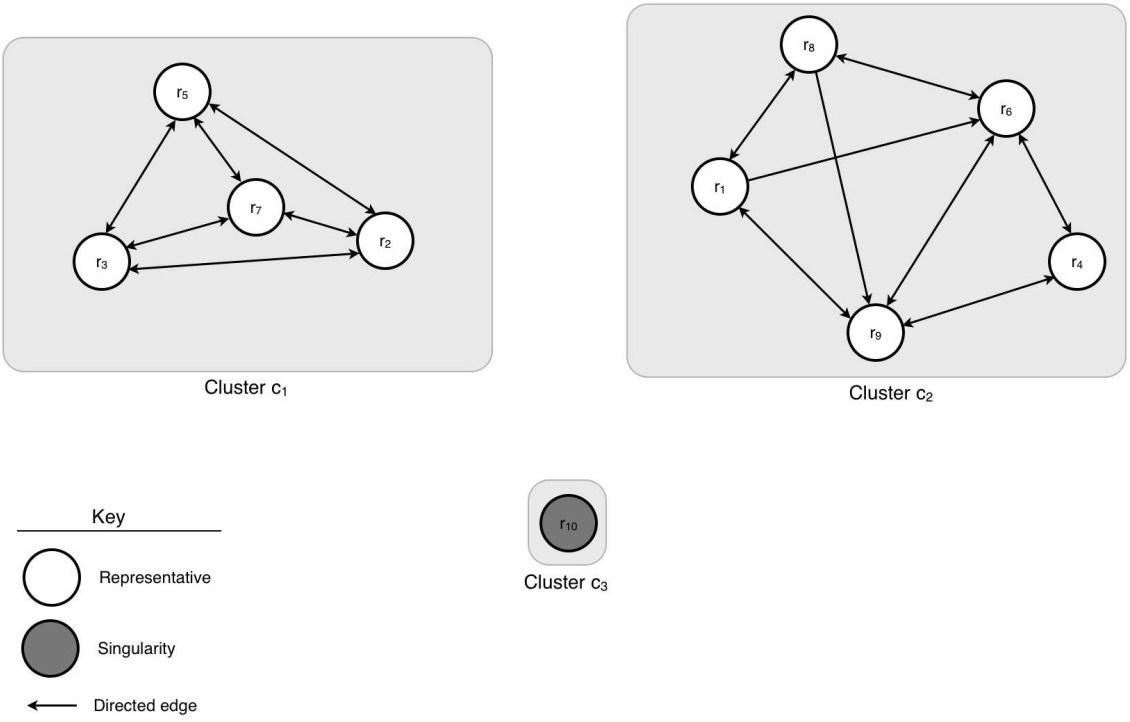


Fig. 4. The representative vertices on sparse graph RSG. The identical vertices $(v_{34} \dots v_{37})$ shown in Figure 1 have created a singularity.

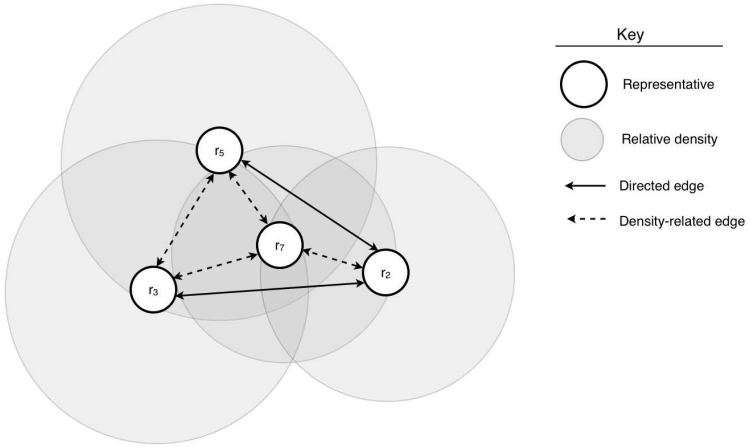


Fig. 5. The relative density of the representative vertices belonging to cluster c_1 when $\alpha = 2$. Reciprocal connections between density-related vertices are highlighted.

Connectivity between representative vertices is further governed by a measure of the relative density of the regions that they represent.

Definition 8 *(relative density) The density of representative vertex $r_i \in R$ on the sparse graph SG is defined by the function $RD(r_i)$:*

$$RD(r_i) = \frac{1}{|NN(r_i)|} \sum_{j=1}^{|NN(r_i)|} D(r_i, NN(r_i, j)) \,. \tag{5}$$

Definition 9 *(density-related) Given a density scaler α, two representatives r_i and r_j are density-related if the following conditions are true:*

$$r_j \in RC_R(r_i) \tag{6}$$

$$D(r_i, r_j) \leq RD(r_i) \cdot \alpha \tag{7}$$

$$D(r_i, r_j) \leq RD(r_j) \cdot \alpha. \tag{8}$$

Definition 10 *(singularity) A representative vertex $r_i \in R$ is termed a singularity when its localised density $RD(r_i)$ has reached zero and the following conditions are met:*

$$\sum_{j=1}^{k} D(r_i, NN(r_i, j)) = 0 \tag{9}$$

$$|NN(r_i)| = k. \tag{10}$$

An example of a singularity is shown as part of the representative sparse graph in Figure 4. The relative densities of the representative vertices in cluster c_1 and the density-relation of the representatives are depicted in Figure 5.

4 Clustering Stream Data via Representative Points

Our cluster representation involves the use of dynamically updated sparse graphs that, when used in conjunction with a repository of representative vertices, allows us to rebuild a cluster's history and to rapidly adapt to significant patterns previously observed. The RepStream algorithm that we propose aims to capture such patterns in order to recall them at some future time should the change reoccur. RepStream is a single phase incremental algorithm that updates two sparse graphs of k-nearest neighbour connected vertices in order to identify clusters among data points. The first of these graphs is used to capture the connectivity relationships amongst the most recently seen data points and to select a set of representative vertices. The second graph is used to track the connectivity between the chosen representative vertices. The connectivity of the representative vertices on both graphs then forms the basis for the algorithm's clustering decision making. An overview of the relationship between the two sparse graphs is provided in Figure 6.

4.1 Algorithm Overview

At each time step a new single point p_i is observed in the data stream and added to the sparse graph $\mathrm{SG}(V, E)$ as vertex v_i. A new vertex joins an existing cluster if it is reciprocally connected to a representative vertex $v_j \in R$. Should no such representative vertex exist then v_i is itself promoted as an exemplar or

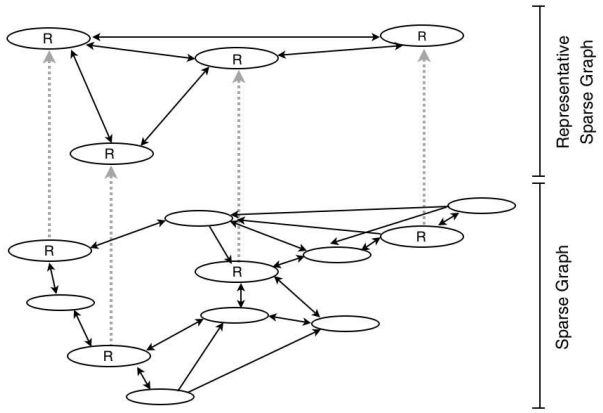


Fig. 6. The relationship between the sparse graph and the representative sparse graph. Representative vertices are labelled "R" while non-representative vertices are unlabelled.

predictor vertex and used to form a new cluster. The creation of the new cluster may trigger an immediate merge with an existing cluster if the conditions for merging are met.

Representative vertices are used to make clustering decisions at a higher level. This offers two major advantages. First, since representative vertices typify a set of nearby data points, decisions made at this level improve performance by requiring only a subset of the data to be considered. Second, representative vertices are associated with a measure of usefulness which allows the algorithm to selectively retain highly valued representatives as historical descriptors of the cluster structures. This retention allows the algorithm to accurately classify new data points arriving within a region of the clustering space where the non-representative vertices have since been retired.

Cluster merges and splits are conditional on the connectivity of representative vertices on the higher level representative sparse graph along with the density of vertices on the sparse graph SG in local proximity to their representative vertices. Since these decisions are made at the representative vertex level, larger clusters are formed when two or more representative vertices are seen to be density-related. Merges can, therefore, be made immediately when an update to the connectivity of vertices on the representative sparse graph sees the creation of a new reciprocal connection between two representative vertices where the localised density of representative vertices suggests that they are of similar density and are in close proximity. Merges may also be triggered when the addition of a new, or the removal of an existing, vertex affects the density of existing representative vertices. Two existing representative vertices that are reciprocally connected but not density-related can merge if the density of one or both representative vertices reduces over time to allow such a connection to be made. Cluster splits are similarly detected by performing a split test whenever existing density-related connections between

two representative vertices are lost. Monitoring the connectivity and relative density of representatives enables the algorithm to evolve with changes in the data.

A significant aim of RepStream is to retain those representative vertices that prove, over time, to be useful in representing the shapes and distributions of clusters. Such vertices are retained for as long as possible (subject to available resources) via a repository. Retaining the most useful representative vertices permits the algorithm to "remember" previously encountered cluster distributions in order to rapidly adapt to recurrent patterns. Those vertices that are no longer useful are archived for possible off-line analysis if constraints prevent the algorithm from retaining them.

Processing and memory constraints require that the algorithm discard information over time. This is accomplished by prioritising the disposal of data such that the least useful information for clustering is removed first. Non-representive vertices are queued on a first in, first out basis and removed whenever resource limitations are reached. Representative vertices that are not stored in the repository are considered to have little retentive value and are also removed via the deletion queue. All other representative vertices remain in memory; their deletion is instead managed via the repository update procedure detailed in Section 4.6.

The removal of a vertex requires updates to the sparse graph and the representative sparse graph. Graph updates are made to ensure that any vertices with edges directed at the removed vertex are updated with a new nearest neighbour. Representative vertices are also updated to ensure that their local density is adequately maintained.

4.2 Sparse Graph SG Updates

Insertion of a new vertex v_i into the sparse graphs is a multistep process that initially links each new vertex into sparse graph SG prior to updating any vertices that have been affected by the insertion. Line numbers referenced here refer to Algorithm 1 which provides an outline of the steps involved. First, a directed edge is created from v_i to each of its nearest neighbours (line 7). A neighbour v_j is updated to link back to v_i if the distance from v_j to the furthest neighbour of v_j is greater than the distance from v_j to v_i (lines 8–9). The creation of a directed edge from v_j to v_i signals the creation of a reciprocal connection between the two vertices and causes vertex v_j to be queued for further processing as a vertex that has gained a reciprocal link (line 10). If the addition of a directed edge from v_j to v_i results in vertex v_j possessing more than k neighbours then the outgoing edge from v_j to its

Algorithm 1: linkIntoGraphSG($v_i, neighbours$)

```
1  createdRecipVertices ← ∅;
2  removedRecipVertices ← ∅;
3  removedRecipRepresentatives ← ∅;
4  removedNeighbours ← ∅;
5  deletedVertex ← ∅;
6  foreach v_j ∈ NN(v_i) do
7      createLink(v_i, v_j, SG);
8      if |NN(v_j)| < k or D(v_i, NN(v_j, |NN(v_j)|)) ≤ D(v_j, NN(v_j, |NN(v_j)|)) then
9          createLink(v_j, v_i, SG);
10         createdRecipVertices += v_j;
11         if |NN(v_j)| > k then
12             remove furthest nearest neighbour of v_j;
13             removedNeighbours += v_j;
14             if removal of furthest neighbour of v_j removes a reciprocal link then
15                 removedRecipVertices += v_j;
16 foreach v_j ∈ createdRecipVertices do
17     if v_j ∈ R then
18         update local density of v_j;
19         deletedVertex = updateRepository(v_j, reinforceCount(v_j) + 1);
20         if deletedVertex then
21             remove deletedVertex from createdRecipVertices, removedRecipVertices,
               removedRecipRepresentatives and removedNeighbours;
22             unlinkVertex(deletedVertex);
23         connectivityChanged = updateRepresentativeStatus(v_j);
24         if connectivityChanged then removedRecipRepresentatives += v_j;
25 if v_i not reciprocally-connected to a representative then makeRepresentative(v_i);
26 else assign v_i to cluster of closest representative;
27 foreach v_j ∈ removedRecipRepresentatives do
28     if v_j ∈ R then
29         update local density of v_j;
30         connectivityChanged = updateRepresentativeStatus(v_j);
31         if connectivityChanged then removedRecipVertices += v_j;
32 foreach v_j ∈ removedNeighbours do
33     foreach v_x ∈ NN(v_j) do
34         if v_x not connected to a representative then makeRepresentative(v_x);
```

furthest neighbour is removed and v_j is added to a queue of vertices that have lost a nearest neighbour (lines 11–13). This ensures that the k-nearest neighbour property of the sparse graph is maintained. Further, if the removal of the furthest neighbour of v_j results in the removal of a reciprocal link then v_j is appended to a queue of vertices that have lost a reciprocal link to a neighbouring vertex (lines 14–15).

The vertices that have gained a reciprocal connection to the new vertex v_i are now revisited (line 16). First, the local density of each representative vertex

reciprocally connected to v_i is updated (line 18). The reinforcement count of reciprocally linked representative vertices is also incremented via the repository update method described in Section 4.6 (line 19). Repository updates may trigger the removal of less useful representative vertices if resource constraints limit the size of the repository (line 20). Such vertices are immediately unlinked from both sparse graph SG and sparse graph RSG, and archived to disk for possible future off-line analysis (line 22). Updates to a representative vertex's local density may cause the removal of a density-related link between two vertices (line 23). If so, the affected vertex is added to a queue of vertices that have lost a reciprocal link to another vertex (line 24) so that cluster split checks may be made. A decision is then made as to whether the new vertex v_i will be promoted to a representative vertex is made. Immediate promotion of a new vertex occurs, if no reciprocal link was established between it and an existing representative vertex, otherwise the new vertex is assigned to the cluster of its nearest representative (lines 25–26).

Non-representative vertices that have lost a reciprocal link are revisited to identify representative vertices that require updates to their local density (lines 27–30). Should such an update trigger the removal of a density-related link then that representative vertex is added to the queue of representatives requiring a cluster split check (line 31). Any non-representative vertices that have lost a nearest neighbour are checked to ensure that its neighbours continue to be reciprocally linked to at least a single representative vertex. Any neighbour that is not reciprocally linked to a representative vertex also becomes a representative vertex (lines 32–34). This ensures that each data point is linked to, or acts as, a representative.

4.3 Sparse Graph RSG Updates

Inserting a new representative vertex into sparse graph RSG involves a process similar to vertex insertions on sparse graph SG. Line numbers here refer to Algorithm 2 which details the steps involved.

As with sparse graph SG, each new representative vertex r_i is first linked to its nearest neighbours on RSG (line 5). A reciprocal link from a nearest neighbour r_j is then made to r_i if the distance from r_j to r_i is less than or equal to the distance of the furthest nearest neighbour of r_j (lines 6–7). The k-nearest neighbourhood property of r_j is then maintained (lines 8–10) which may trigger the removal of a reciprocal link (lines 11–12).

The representative vertices are now used to detect cluster merges, the trigger condition being the creation of a density-related link between v_i and a representative vertex that has been assigned to another cluster (line 13). Finally,

Algorithm 2: linkIntoGraphRSG($r_i, neighbours$)

```
1  createdRecipVertices ← ∅;
2  removedRecipRepresentatives ← ∅;
3  removedNeighbours ← ∅;
4  foreach r_j ∈ NN_R(v_i) do
5      createLink(r_i, r_j, RSG);
6      if |NN_R(r_j)| < k or D(r_i, NN_R(r_j, |NN_R(r_j)|)) ≤ D(r_j, NN_R(r_j, |NN_R(r_j)|))
       then
7          createLink(r_j, r_i, RSG);
8          createdRecipVertices += r_j;
9          if |NN_R(r_j)| > k then
10             remove furthest nearest neighbour of r_j;
11             if removal of furthest neighbour of r_j removes a reciprocal link then
12                 removedRecipRepresentatives += r_j;
13     if cluster of r_i ≠ cluster of r_j then merge clusters of r_i and r_j;
14 foreach vertex r_j ∈ removedRecipRepresentatives do splitCheck(r_j);
```

split checks are performed on those representative vertices that have lost a reciprocal link to another vertex on RSG (line 14).

In terms of complexity, in order for a new representative r_i to be inserted into the sparse graph RSG, it is required that the representative's nearest k-nearest neighbours be initially identified which involves a $\mathcal{O}(k \log |R|)$ operation when employing the KD-Tree. Creating an edge from r_i to a single neighbour is accomplished in constant time. Maintaining the ordering of a the nearest neighbours of r_i, however, requires an additional $\mathcal{O}(\log k)$ operation. Next, the algorithm checks the distance of r_i to each of its neighbours to test whether the neighbouring representatives need to update their own nearest neighbourhood. This requires distance calculations of complexity $\mathcal{O}(d)$ for each of its neighbours. If each of the nearest neighbours of r_i becomes reciprocally linked to r_i then this introduces additional $\mathcal{O}(\log k)$ processing to establish an reciprocal link from a neighbour r_j back to r_i. The removal of the furthest nearest neighbour of r_j is tracked in constant time.

The worst case complexity of updating the links between r_i and all of its k neighbours is therefore

$$\mathcal{O}(k \log |R|) + \mathcal{O}(2k \log k) + \mathcal{O}(kd)\,. \tag{11}$$

An example of the sparse graph updates made due to the arrival of a new vertex is shown in Figure 7. Here the existing sparse graph in Figure 7a is updated with a new vertex with the updated edges shown in Figure 7b. The new vertex is not reciprocally linked to an existing representative vertex and, as such, is promoted as a new representative as seen in Figure 7c.

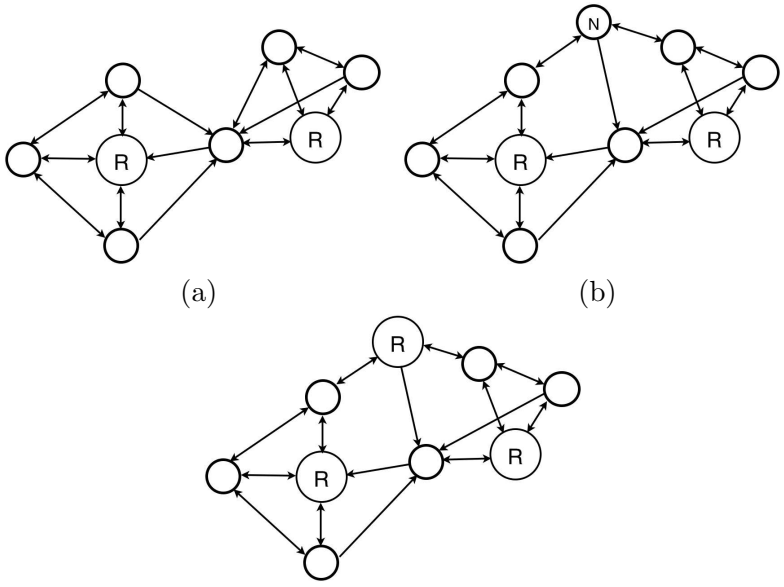


Fig. 7. Incremental sparse graph update: the connectivity of vertices in the cluster in (a) is updated with the arrival of a new vertex "N" in (b). The new vertex is not reciprocally connected to any representative vertex "R" and is therefore made representative in (c).

RepStream performs periodic vertex removals. This process requires updates being made to each vertex connected to the vertex v_i being removed. A total of k outgoing edges and $|\mathrm{IE}(v_i)|$ incoming edges (Definition 2) are removed from the sparse graph SG. Further, if the vertex being unlinked is a representative then a further k outgoing links and $|\mathrm{IE}_{\mathrm{R}}(v_i)|$ incoming edges will be removed from the sparse graph RSG. The complexity of these edge removals is $\mathcal{O}(2k)$ + $\mathcal{O}(|\mathrm{IE}(v_i)|)$ + $\mathcal{O}(|\mathrm{IE}_{\mathrm{R}}(v_i)|)$.

Each vertex that was connected to the deleted vertex must be revisited to maintain its nearest neighbourhood. Each revisited vertex requires a $\mathcal{O}(k \log |V|)$ search to be performed for each of its k nearest neighbours. The complexity of maintaining the nearest neighbours of each neighbour of v_i is hence $\mathcal{O}(k^2 \log |V|)$. Maintaining the nearest neighbour of the affected representatives on RSG is similarly $\mathcal{O}(k^2 \log r)$. The worst case complexity of removing a single vertex from both sparse graphs is therefore

$$\mathcal{O}(2k) + \mathcal{O}\Big(k^2 \log |V|\Big) + \mathcal{O}\Big(k^2 \log |R|\Big)\,. \tag{12}$$

The initial process of linking a vertex v_i into sparse graph SG in Algorithm 1 is similar to that of Algorithm 2. The complexity of updating the links between v_i and all of its k neighbours is therefore similar to that of Equation 11:

$$\mathcal{O}(k \log |V|) + \mathcal{O}(2k \log k) + \mathcal{O}(kd)\,. \tag{13}$$

The complexity of the remaining operations is dependant on whether a reciprocal link has been established to a representative vertex. If not then the new vertex is made into a representative, the complexity of which is given as Equation 16.

If, however, a reciprocal link to a representative is established then processing continues as follows. First, updating the local density of each representative is done in constant time since summary information can be maintained. Repository updates are then made, the complexity of which will be shown in Section 4.4 to be $\mathcal{O}(3 \log |S|)$. The cost of unlinking a vertex was previously given as Equation 12. Checking the link status of the representative vertices costs $\mathcal{O}(2kd)$ time. The worst case sparse graph SG insertion occurs when each k neighbour of a newly inserted vertex is found to be reciprocally connected to a neighbour. In this case an additional complexity is incurred:

$$\mathcal{O}(3k \log |S|) + \mathcal{O}\Big(2k^2\Big) + \mathcal{O}\Big(k^3 \log |V|\Big) + \mathcal{O}\Big(k^3 \log |R|\Big) + \mathcal{O}\Big(2k^2 d\Big) \quad (14)$$

The complexity of creating a representative vertex will be shown in Section 4.4 to be $\mathcal{O}(k \log |R|) + \mathcal{O}(2k \log k) + \mathcal{O}(3kd)$ (Equation 16). Creating a representative vertex out of each neighbour of the neighbouring vertices of v_i therefore requires

$$\mathcal{O}\Big(k^2 \log |R|\Big) + \mathcal{O}\Big(2k^2 \log k\Big) + \mathcal{O}\Big(3k^2 d\Big) + \mathcal{O}\left(k^2 \frac{|V|}{2}\right) + \mathcal{O}\Big(k^2 |R|^2\Big) \,. \quad (15)$$

Our implementation of RepStream employs a KD-Tree to perform nearest neighbour searches. Although identifying k neighbours incurs, on a naive KD-Tree implementation $\mathcal{O}(k \log k)$, this structure does introduce some additional overhead in order to maintain a balanced structure. Cost of rebuild is approximately $\mathcal{O}\Big(|V| \log^2 |V|\Big)$ on the sparse graph SG, a cost which is amortised over time by only periodically rebuilding the tree.

4.4 Representative Vertex Creation and Promotion

A vertex v_i becomes a representative vertex if at any time it is not reciprocally linked to an existing representative vertex. Algorithm 3 outlines the steps taken to promote a vertex to a representative. A new *representative vertex* is classified as a *predictor* if its number of incoming edges is beneath a threshold that we define to be half of a vertex's maximum outgoing edge count as given by the user supplied value of k (lines 2–7). A *representative vertex* is therefore a *predictor* if $|\mathrm{IE}_{\mathrm{R}}(r_i)| < \frac{k}{2}$ else it is labelled an *exemplar*.

Algorithm 3: makeRepresentative(v_i)

```
1  R = R ∪ v_i;
2  if |IE_R(v_i)| ≥ k/2 then
3  |  mark v_i as an exemplar;
4  |  R^E = R^E ∪ v_i;
5  else
6  |  mark v_i as a predictor;
7  |  R^P = R^P ∪ v_i;
8  if v_i not assigned to a cluster then
9  |  create a new cluster c_new;
10 |  assign v_i to c_new;
11 linkIntoGraphRSG(v_i, NN_R(v_i));
12 foreach e_p ∈ OE(v_i) do
13 |  if term(e_p) ∉ R and D(v_i, term(e_p)) < D(v_i, closest representative of e_p) then
14 |  |  set v_i as closest representative of term(e_p);
15 |  |  relabel term(e_p) and move into assigned cluster of v_i;
```

The initial steps in the creation of a new representative r_i in Algorithm 3 require constant time operations. Testing whether the nearest representative of a single neighbour of r_i has changed requires two $\mathcal{O}(d)$ distance calculations. The worst case complexity of creating a new representative is therefore

$$\mathcal{O}(k \log |R|) + \mathcal{O}(2k \log k) + \mathcal{O}(3kd)\,. \tag{16}$$

A vertex that is becoming representative will not have been assigned to a cluster if it is still in the process of being inserted into the sparse graph at the time of its promotion. In this case, a new cluster is created such that the new representative vertex is the cluster's sole member (lines 8–10). This approach permits the formation of new clusters when the arrival of a new data point falls outside the region of an existing cluster. The new representative vertex is then inserted into the representative sparse graph using the described graph insertion method, possibly triggering a merge between its cluster and an existing cluster (line 11). Figure 8 demonstrates the formation of a new cluster

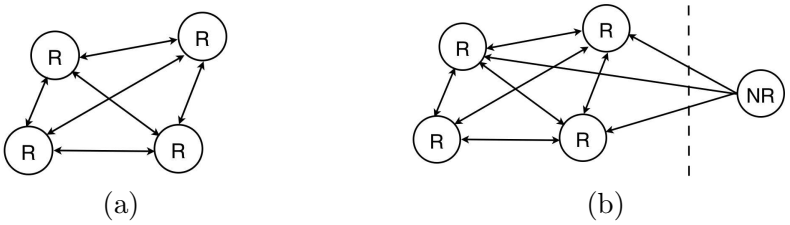


Fig. 8. Formation of a new cluster: the representative sparse graph in (a) is updated with a new representative vertex in (b). The new representative "NR" forms a new cluster due to a lack of reciprocal links with a representative of the existing cluster. The dotted line marks the cluster separation.

Algorithm 4: updateRepresentativeStatus(r_i)

```
1  if r_i ∈ R^P and |IE_R(r_i)| ≥ k/2 then
2      upgrade r_i to an exemplar;
3  statusChanged ← false;
4  foreach e_p ∈ OE_R(r_i) do
5      if e_p is density-related then
6          if D(r_i, term(e_p)) > RD(r_i) · α or D(r_i, term(e_p)) > RD(term(e_p)) · α then
7              mark e_p as reciprocal;
8              statusChanged ← true;
9          else if e_p is reciprocal then
10             if D(r_i, term(e_p)) ≤ RD(r_i) · α and D(r_i, term(e_p)) ≤ RD(term(e_p)) · α
               then
11                 mark e_p as density-related;
12                 statusChanged = true;
13 return statusChanged
```

due to the creation of a new representative vertex that is not reciprocally connected to the existing cluster.

The cluster membership of the nearest neighbours of v_i on sparse graph SG are checked following the insertion of the new representative vertex into the representative sparse graph. A nearest neighbour is relabelled if the new representative vertex is its nearest representative and the cluster membership of the two vertices differ (lines 12–15).

The density of representative vertices is affected over time by changes in the distribution of vertices on sparse graph SG. Reciprocal links between vertices on the representative sparse graph may, therefore, need to be reclassified as being either simple reciprocal or density-related links. This is a necessary step that enables the algorithm to evolve as the density and distribution of the data changes. Algorithm 4 details the steps involved.

Limitations in available memory requires the algorithm to discard old information. A vertex v_i that is to be retired must first be unlinked from its respective graphs in order to maintain nearest neighbourhood connectivity. This requires that each neighbouring vertex of v_i be visited and its nearest neighbours updated.

In terms of complexity, the first steps, testing a predictor representative and possibly promoting it to an exemplar, are achieved in constant time. Checking whether a density-related relationship has been lost requires two $\mathcal{O}(d)$ distance calculations. Two $\mathcal{O}(d)$ calculations are also required to test a link to a non-density-related neighbour. The worst case complexity of checking a representative's status is therefore $\mathcal{O}(2kd)$, regardless of their density-related status.

Cluster split and merge decisions are made by monitoring both the reciprocal connectivity of vertices on the representative sparse graph as well as their relative density based on the proximity of their nearest neighbours on sparse graph SG. The trigger condition for either of these events is the creation or removal of *density-related* links.

Cluster merges are triggered when updates to the representative sparse graph creates a density-related link between representative vertices that have been assigned to different clusters. Merging is a simple process that sees the vertices from the smaller of the two merging clusters moved over to the larger cluster structure. Neither the sparse graph nor the representative sparse graph require any further updating.

Cluster reassignment during a cluster merge is carried out on the smaller of any two clusters being merged. The worst case scenario results in a $\mathcal{O}\left(\frac{|V|}{2}\right)$ cluster merge each time a density-related reciprocal link is established on RSG.

An example of a cluster merge, due to the creation of a new representative that has created a density-related link between two closely positioned clusters, is given in Figure 9.

Split checks are executed when the loss of a density-related link between two vertices on the representative sparse graph is detected. Figure 10 provides an example of a cluster split due to a change in the density-related connections of representative vertices. The change here is caused by the addition of a new representative whose inclusion has removed a reciprocal connection between two representative vertices. An example of a cluster split due to a change in the density of a representative vertex is shown in Figure 11. A standard $O(n^2)$ region growing algorithm that follows the density-related links of the representative vertices was employed to perform split checks.

Split tests execute with complexity $\mathcal{O}(|R|^2)$ given that a simple region growing

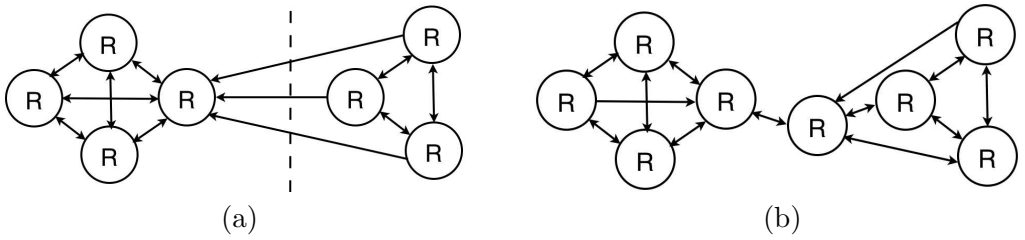
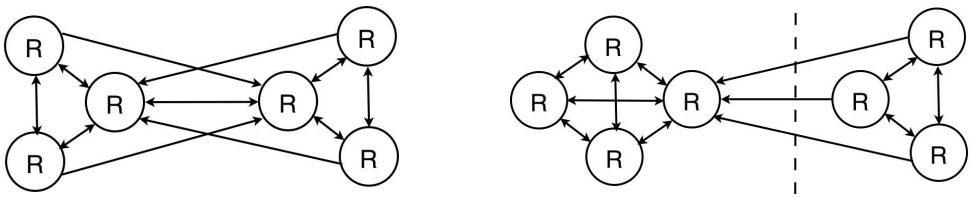
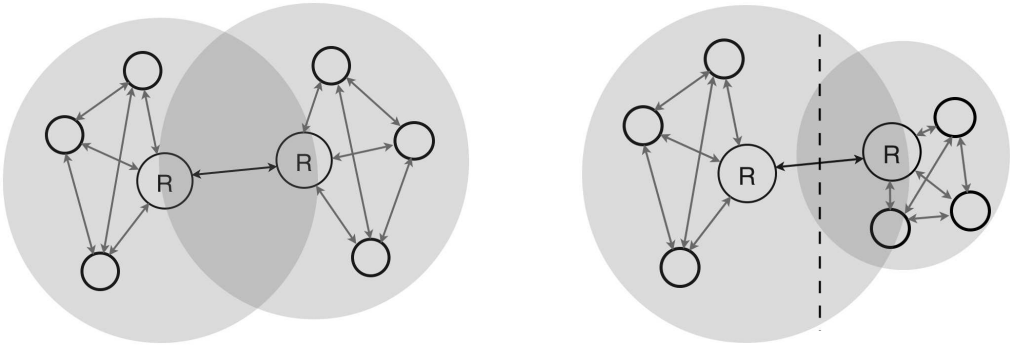


Fig. 9. Cluster merge: the representative sparse graph in (a) is updated with a new representative that causes the two clusters to merge (b). The dotted line marks the cluster separation; areas of influence of each representative vertex's localised density are not shown.



(a) (b)

Fig. 10. Cluster split: the representative sparse graph in (a) is updated with a new representative that causes a split in (b). The dotted line marks the cluster separation; areas of influence representative of each representative vertex's localised density are not shown.



(a) (b)

Fig. 11. Changing density: two representative vertices density-related in (a) split into two separate clusters over time as the density of the second representative increases in (b). Although the representatives remain reciprocally linked, the shaded area of influence centred around the second representative has shrunk. Edges are depicted for both the sparse graph SG and the representative sparse graph. The dotted line marks the cluster separation.

algorithm is used in our implementation.

4.6 Repository Management

Central to the operation of RepStream is a repository of the most useful representative vertices used for clustering.

Definition 11 *(representative usefulness) The usefulness of a representative vertex r_i is defined by the decay function:*

$$\begin{aligned} usefulness\,(r_i, count) = &\log(\lambda) \cdot (current\ time - creationTime\,(r_i) + 1) \\ &+ \log(count + 1) \end{aligned} \tag{17}$$

where λ is a user specified decay rate and count is the representative vertex's reinforcement count.

The repository is defined as an ordered vector of vertices $S = \left\langle s_1, \ldots s_{|S|} \right\rangle$ sorted in ascending usefulness. The decay function ensures a monotonic ordering of vertices in the repository with respect to the passing of time. In our implementation of RepStream we chose to index the repository using an AVL binary search tree [42]. Algorithm 5 outlines the repository update process. Updating the reinforcement count of a representative vertex that has already been added to the repository requires only two tree operations: the removal of the vertex and then its subsequent reinsertion following an increment to its reinforcement count (lines 2–4). The least useful representative vertex can be rapidly found by traversing to the AVL tree node with the lowest usefulness score.

Approximately 50% of the memory allocated to the algorithm is reserved for maintaining the representative vertex repository. This value was chosen to balance the algorithm's ability to retain historical information with its capacity to discover slow forming clusters. Such clusters may not be evident if only the short term behaviour of the data stream were to be analysed.

New additions to the repository are made whenever a new representative vertex is created until resource constraints have been reached (lines 6–7). At this point only the most useful repository members are retained. This is achieved by comparing the least useful repository member with other non-repository representatives whenever their reinforcement count is incremented (lines 9–13).

The process of updating an existing repository member's reinforcement count outlined in Algorithm 5 requires a single removal and a single insertion operation, each of which can be performed in $\mathcal{O}(\log |S|)$ time using an AVL tree [42].

Algorithm 5: updateRepository $(v_i, newRCount)$

```
1  deletedVertex ← ∅;
2  if r_i ∈ S then
3  |  remove r_i from S;
4  |  insert r_i into S with newRCount;
5  else
6  |  if |S| < maxRepositorySize then
7  |  |  insert r_i into S with newRCount;
8  |  else
9  |  |  r_w ← first(S);
10 |  |  if usefulness(r_w, newRCount) ≤ usefulness(r_i, newRCount) then
11 |  |  |  remove r_w from S;
12 |  |  |  insert r_i into S with newRCount;
13 |  |  |  deletedVertex ← r_w;
14 set reinforcement count of r_i to newRCount;
15 return deletedVertex
```

If a new member is being inserted into the repository and an old entry is removed then an additional $\mathcal{O}(\log |S|)$ search operation is required to identify the least useful member. The worst case complexity of a repository update is therefore $\mathcal{O}(3 \log |S|)$.

An example of how the repository retains the shape of a cluster in the presence of recurrent change is demonstrated in Figure 12.

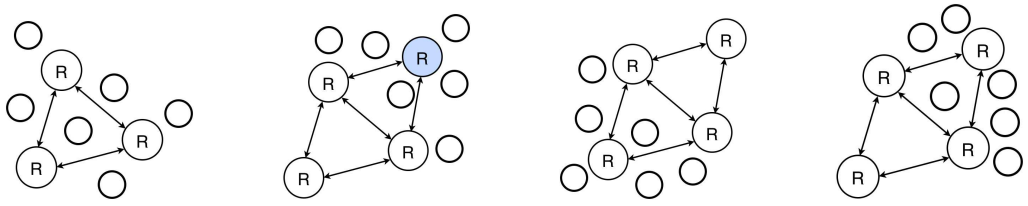


Fig. 12. Recurrent change: the initial cluster with representatives "R" is shown in (a). Change is evident in the cluster as new vertices arrive and old vertices are removed in (b). This change results in the creation of a new representative vertex (shown shaded). The recurrent pattern in (c) reinforces the existing representatives; the newly formed representative is retained. The new representative vertex is then reinforced with the recurrent change in (d). For clarity, edges between non-representatives are not shown.

4.7 Singularities

A representative vertex r_i that is k-connected to its nearest neighbours and where $\sum_{j=1}^{k} \mathrm{D}(r_i, \mathrm{NN}(r_i, j)) = 0$ represents a collection of identical points that offer no new information to the clustering process, yet whose inclusion in the sparse graphs would require the retirement of otherwise useful vertices. For example, a stationary object in a GPS data stream is likely to introduce a large number of identical points that are well suited for representation by a single representative vertex.

Such vertices are referred to as *singularities* and occur when identically featured points are frequently observed in a data stream. New points that are identical to a singularity can therefore be immediately deleted in order to avoid the overhead of unnecessary sparse graph updates and to maintain the information value of the repository. The occurrence of such points is not lost, however, and is instead represented through the singularity's reinforcement count.

Singularities are unable to be assigned non-zero density measures and as such do not lose their singularity status once it is acquired. This ensures that the presence of a singularity is permanently captured by the algorithm even though its nearest neighbours may be retired over time. Representative vertices are unable to form density-related links to singularity vertices.

5 Experimental Results

The performance of RepStream [1] was evaluated using both synthetic and real world data sets. Our real world data sets consisted of the KDD-99 Cup network intrusion data [2] and the forest canopy type data [3] first described in [16]. Although the latter data set cannot be regarded as true stream data, we make the assumption that the ordering of the data can be used as a substitute for time. The forest data set was previously treated as stream data in [5].

Both cluster purity [3] and cluster entropy [33] were used to measure the classification accuracy of the algorithm.

Cluster purity provides a measure of how well data is being classified in the short term over a horizon of the previous h data points. The cluster purity $\mathrm{CP}(c_i)$ of a cluster c_i is defined as

$$\mathrm{CP}\left(c_i\right) = \frac{1}{|c_i|} \max_k \left(\sum_{j=1}^{|c_i|} \mathrm{r}\left(v_j, k\right) \right)$$

where $\mathrm{r}\left(v_j, k\right)$ is the counting function:

$$\mathrm{r}\left(v_j, k\right) = \begin{cases} 1, & \text{if class}\left(v_j\right) = k \\ 0, & \text{otherwise.} \end{cases}$$

The total clustering purity $\mathrm{TCP}(C)$ is then found by averaging over all clusters:

$$\mathrm{TCP}\left(C\right) = \frac{1}{|C|} \sum_{i=1}^{|C|} \mathrm{CP}\left(c_i\right). \tag{18}$$

As well as measuring the purity of the clustering, cluster entropy was employed as an indicator of the homogeneity of the entire cluster system. The entropy of a cluster c_i is defined as

$$\mathrm{H}\left(c_i\right) = \sum_{k=1}^{\text{classes in } c_i} \left| \mathrm{q}\left(c_i, k\right) \times \log\left(\mathrm{q}\left(c_i, k\right)\right) \right| \tag{19}$$

[1] Available from http://impca.cs.curtin.edu.au/downloads/software.php
[2] Available [11] from http://kdd.ics.uci.edu/databases/kddcup99/
[3] Available [34] from ftp://ftp.ics.uci.edu/pub/machine-learning-databases/covtype/

given the counting function

$$\mathrm{q}\left(c_i, k\right) = \frac{\sum_{j=1}^{|c_i|} \mathrm{r}\left(v_j, k\right)}{|c_i|}. \tag{20}$$

The total cluster entropy can thus be found via

$$\mathrm{TH}\left(C\right) = \sum_{i=1}^{|C|} \frac{|c_i|}{\sum_{j=1}^{|C|}\left(|c_j|\right)} \times \mathrm{H}\left(c_i\right). \tag{21}$$

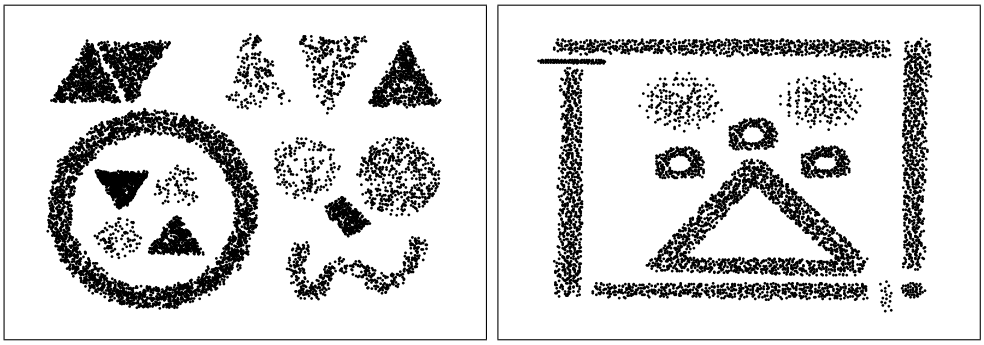
Unless noted otherwise, the algorithm was constrained to using only 10 MiB of memory and the decay factor used in all experiments was set to $\lambda = 0.99$. The ordering of the data sets was invariant throughout the experimentation. The KD-Tree [14,45] was used to perform nearest neighbour searches in our implementation.

5.1 Synthetic Data

The clustering quality of RepStream was first examined using two hand crafted synthetic data sets. The quality of RepStream was compared against an incremental version of DBSCAN [23]. DBSCAN was selected for comparison as it employs a density based method of clustering known to perform well with arbitrarily shaped clusters. The algorithm is, however, limited to operating at a single density and is therefore expected to exhibit difficulties when dealing with the synthetic data sets. As DBSCAN relies on a priori knowledge of the optimal cluster density, we repeated each of the DBSCAN experiments using a variety of values for the ϵ-neighbourhood. The minimum number of points required to form a cluster was set to 5. The experiment was designed to demonstrate that RepStream is capable of clustering a difficult set containing a variety of arbitrarily shaped clusters of different densities without requiring specialised expert assistance to select the algorithm parameters.

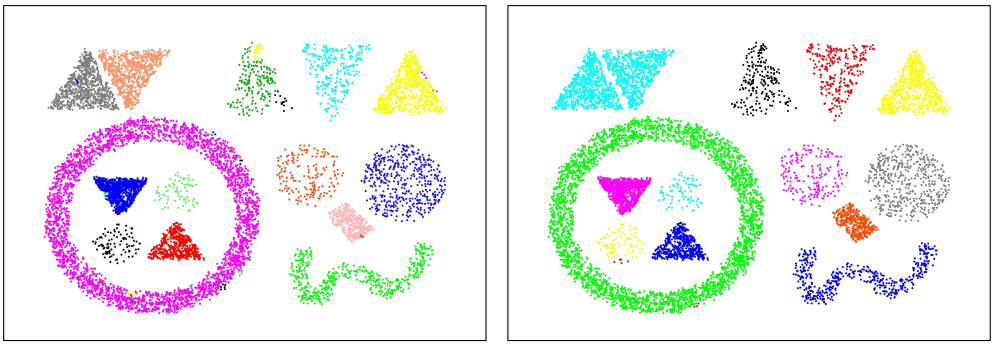
The two data sets used, DS1 and DS2, are presented in Figure 13. Both data sets were presented to the two algorithms using a randomised point ordering that did not change during experimentation. The similarity between points was computed using the Manhattan distance. DS1 was made up of 9,153 data points while DS2 consisted of 5,458 points.

Figure 14a depicts the RepStream clustering of DS1 using the optimal parameter set $k = 4$ and $\alpha = 4.0$. These results show that the algorithm was able to cluster the arbitrarily shaped clusters well. The discovered clusters are



(a) (b)

Fig. 13. Synthetic datasets (a) DS1 and (b) DS2.



(a) (b)

Fig. 14. RepStream clustering of DS1 highlighting the performance difference between a neighbourhood connectivity of (a) $k = 4$ and (b) $k = 5$ when $\alpha = 4.0$. The higher connectivity in (b) resulted in the removal of the fragmentation of the middle triangle at the cost of an incorrect merger between the two upper left corner triangles.

sub-optimal, however, with some minor fragmentation of clusters evident. The most notable of these is seen in the middle triangle at the top of the cluster set, a close up of which is shown in Figure 15. The figure shows that the algorithm has identified two separate clusters within the overall triangular cluster; the separate clustering of these points is not considered an error, however, as their location and density suggests that these points may, indeed, belong to separate clusters when compared to the remaining points.

Increasing the density scaler from $\alpha = 4.0$ to a higher value of $\alpha = 6.0$ did not correct this clustering. Decreasing the scaler did, however, result in increasingly fragmented clustering. An increase of the neighbourhood connectivity successfully overcame the fragmentation issue of the cluster in Figure 15. The result, depicted in Figure 14b, shows that the correction of the fragmentation has come at a price, however, with the incorrect merging of the closely spaced top two left triangular clusters.

In contrast, the single density approach of DBSCAN was found to produce

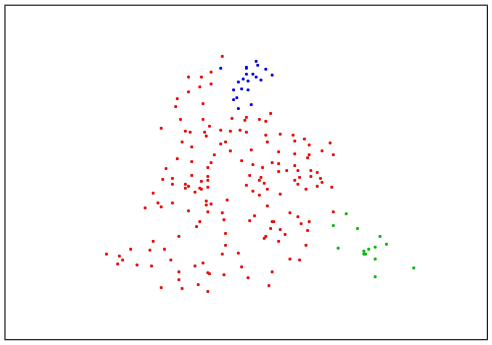
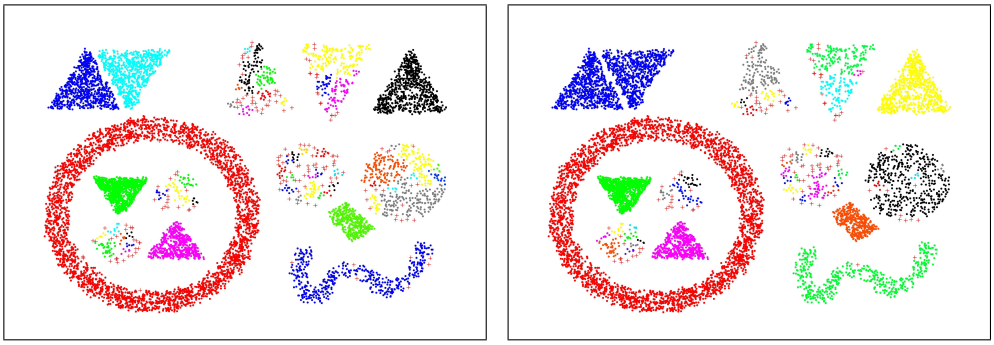


Fig. 15. Close up view of the triangle in DS1 with fragmented clustering by RepStream with $k = 4$ and $\alpha = 4.0$.



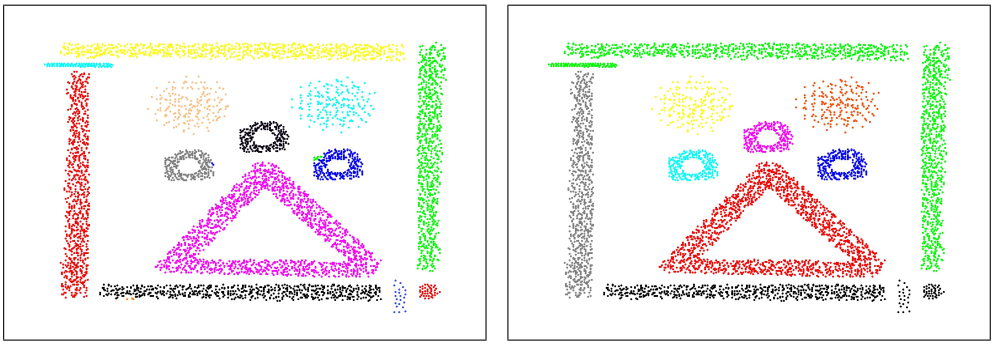
(a) (b)

Fig. 16. DBSCAN clustering of DS1 showing (a) significant cluster fragmentation and unclustered points when $\epsilon = 15$ and (b) the introduction of two incorrectly merged clusters when $\epsilon = 16$.

well formed higher density clusters with an ϵ-neighbourhood parameter of $\epsilon = 15$. The lower density clusters of DS1, however, were found to be highly fragmented with the presence of a significant number of unclustered points that the algorithm treated as noise. These results are shown in Figure 16a. Decreasing the density with $\epsilon = 16$ marginally decreased the cluster fragmentation, as seen in Figure 16b, though at the expense of the incorrect merging of the two top left triangular clusters.

Similar results were obtained clustering data set DS2. The RepStream produced clusters with parameters $k = 4$ and $\alpha = 3.0$ are given in Figure 17a. Again, the algorithm has discovered the clusters with only some minor fragmentation visible in two of the clusters that was not able to be overcome by further increasing the value of α. Lower values did, however, introduce increasing amounts of further fragmentation. Incrementing the neighbourhood connectivity to $k = 5$ overcame the fragmentation issue although this did once again cause several nearby clusters to be incorrectly merged as seen in Figure 17b.

Figure 18 shows the result of clustering DS2 with DBSCAN with a selection of ϵ values. The first of these, $\epsilon = 6$ in Figure 18a, shows that although no clusters



(a) (b)

Fig. 17. RepStream clustering of DS2 showing (a) good clustering with a neighbourhood connectivity of $k = 4$ and density scaler of $\alpha = 3.0$ and, (b) incorrect merging of several closely spaced clusters when the connectivity was increased to $k = 5$.

have been incorrectly merged, a significant number of points have remained unclustered. Results from using a slightly higher ϵ-neighbourhood value of $\epsilon = 7$ in Figure 18c shows that the issue of the unclustered points remains and has also introduced an incorrect merge between two closely positioned clusters. Although unclustered points still remain, optimal clustering was achieved with $\epsilon = 6.5$ as seen in Figure 18b.

This demonstrates the difficulty of selecting an optimal density value without prior knowledge of the *final* data distribution in the synthetic data sets. Stream data compounds this problem as the absolute minimum density is a sensitive measure that is potentially undergoing constant change in a data stream. Finding optimal values of k and α for RepStream is easier as these parameters tend to affect the relative relationships between points rather than introducing static thresholds. The parameter sensitivity of RepStream is further discussed in Section 5.4.

5.2 Network Intrusion Data

The KDD Cup-99 data set features network connection data derived [50] from seven weeks of raw TCP logs consisting of both regular network traffic as well as 24 types of simulated attacks within a military local area network. The data is available both as a complete set that contains approximately 4.9 million records and as a 10% sub-sampled set containing 494,020 points. Each connection record consists of 41 features plus a class ID. Of the available dimensions, 34 continuous valued features were used for clustering and a single outlier point was removed. Accurate clustering of this data demonstrates that the algorithm is able to cope in real world situations where a data stream periodically contains bursts of unexpected and unusual data records.

RepStream was tested on the sub-sampled data set using both a purity hori-

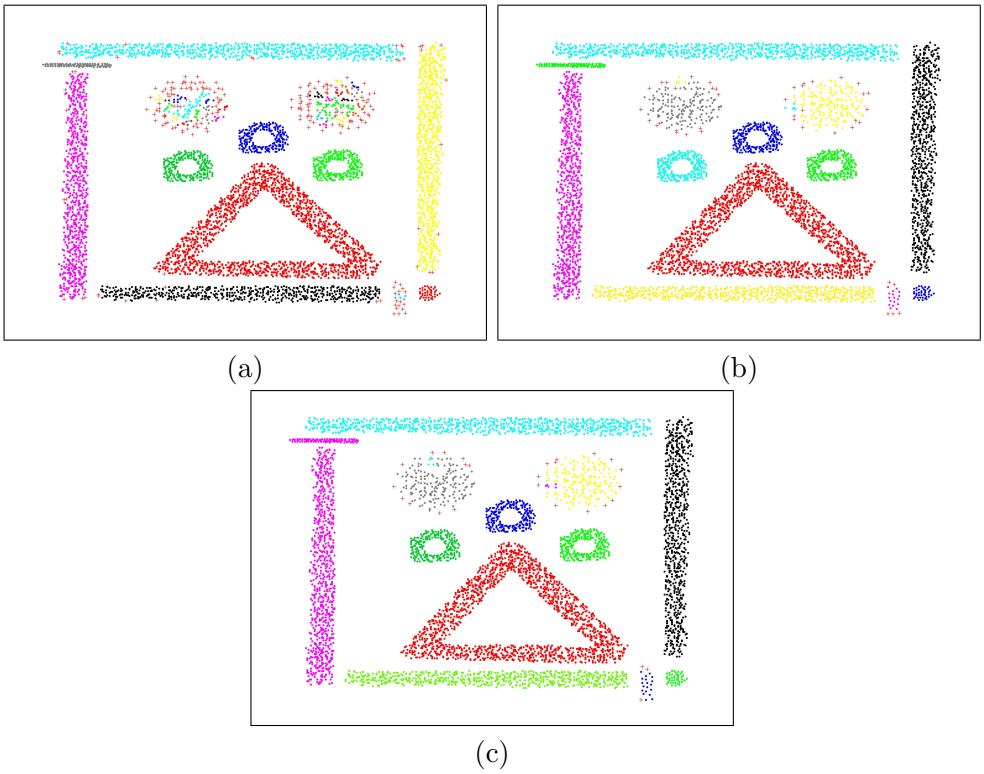


Fig. 18. DBSCAN clustering of DS2 showing (a) a lack of cluster formation with $\epsilon = 6$, (b) optimal clustering with $\epsilon = 6.5$ and (c) the incorrect merger of two closely spaced clusters when $\epsilon = 7$. Unclustered points are indicated by red crosses.

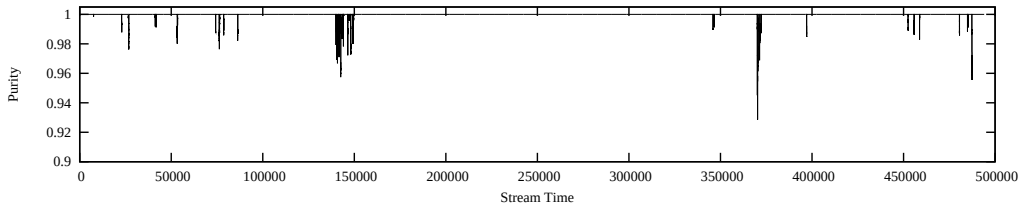
zon of length $h = 200$ and a horizon of length $h = 1,000$, two common horizon lengths used in previous work to evaluate clustering accuracy in data streams [4,5].

The Manhattan distance function was used to compute the similarity of data points from features that were normalised on-the-fly. A point $p_i = \{p_{i,1} \dots p_{i,D}\}$ of D dimensions was normalised in each dimension d using the formula:

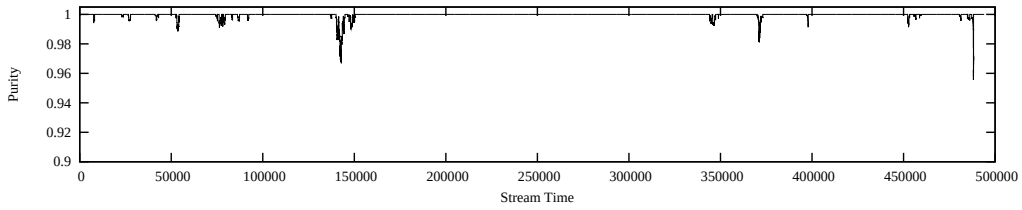
$$p'_{i,d} = \frac{p_{i,d}}{\sum_{j=1}^{|P|} p_{j,d}} \tag{22}$$

where $|P|$ refers to the number of points in memory at any given time. The nearest neighbourhood connectivity was set to $k = 5$ and a density scaler of $\alpha = 1.5$ was employed.

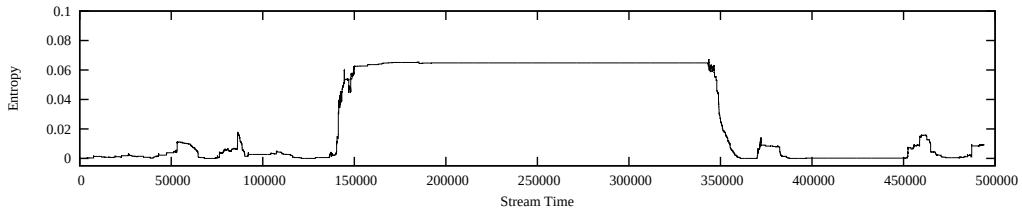
The results of the purity experiment are given in Figure 19a and Figure 19b for $h = 200$ and $h = 1,000$ respectively. These plots, along with the entropy shown in Figure 19c, show that RepStream is able to accurately differentiate between different types of attack connections.



(a)

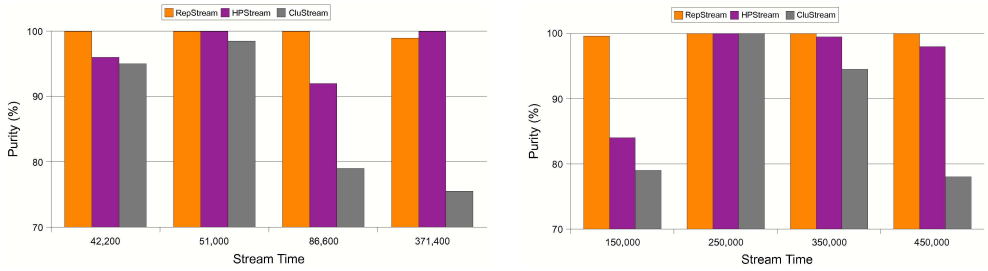


(b)



(c)

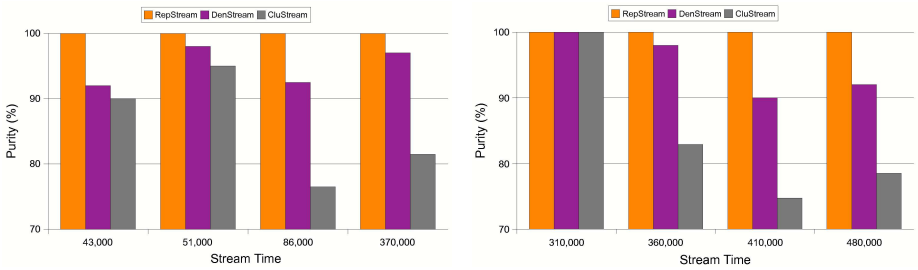
Fig. 19. Purity measure throughout the network intrusion data stream with horizon of (a) $h = 200$ points and (b) $h = 1,000$ points and (c) the cluster entropy when $k = 5$ and $\alpha = 1.5$.



(a) (b)

Fig. 20. Comparative purity measures of RepStream, HPStream and CluStream for (a) $h = 200$ and (b) $h = 1,000$ using available published results on the sub-sampled KDD Cup 1999 data set. The RepStream parameters were $k = 5$ and $\alpha = 1.5$. Stream sample times match those reported in [4,5].

The accuracy of RepStream was also evaluated against published results reported on the same data set for the HPStream, DenStream and CluStream algorithms. The results of the comparisons, depicted in Figure 20 and in Figure 21, show that in most cases RepStream was able to classify network connections as well as or with higher accuracy than HPStream, DenStream and CluStream. The data stream sample times were chosen to match those reported in [4,5].



(a) (b)

Fig. 21. Comparative purity measures of RepStream, DenStream and CluStream for (a) $h = 200$ and (b) $h = 1,000$ using available published results on the sub-sampled KDD Cup 1999 data set. The RepStream parameters were $k = 5$ and $\alpha = 1.5$. Stream sample times match those reported in [4,5].

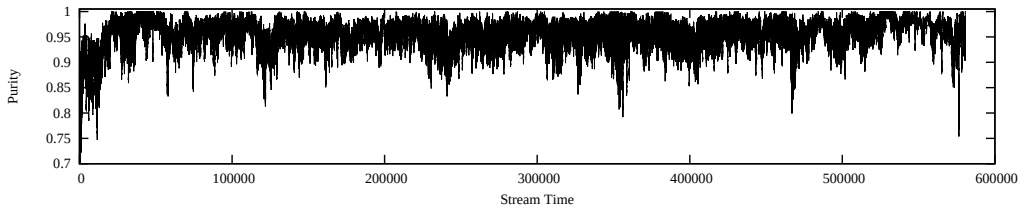
5.3 Forest Cover Data

The forest cover data set contained 581,012 records consisting of a total of 54 geological and geographical features that describe the environment in which trees were observed. Records also included a class ID providing a ground truth as to which of seven different types of canopy were present on the trees. Attributes consisted of a mixture of continuous valued and Boolean valued data, the latter taking values from the set $\{0, 1\}$. Successful clustering of this data set will demonstrate that the algorithm is able to cope with a highly dynamic data set when compared to the network intrusion experiment in Section 5.2.

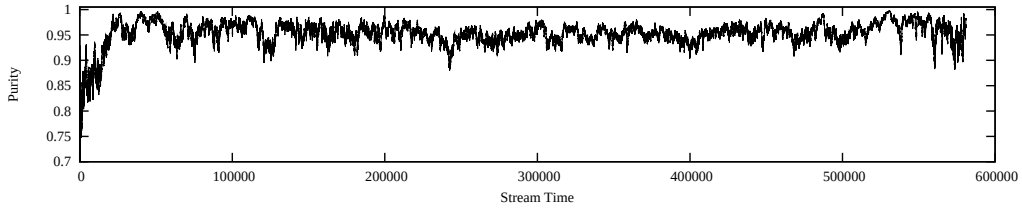
Dimensions were normalised on-the-fly as described in Section 5.2 and the Manhattan distance function was used to measure the similarity between points. The RepStream parameters used on this data set were $k = 5$ and $\alpha = 1.5$.

Figure 22a and Figure 22b show the purity measured over the data stream for $h = 200$ and $h = 1,000$ purity horizons. The plots show that RepStream was able to classify the canopy types with an accuracy typically $\geq 90\%$ over a purity horizon of 1,000 points and typically $\geq 85\%$ on the 200 point horizon. The jagged appearance of the purity plots along with the entropy measure depicted in Figure 22c is expected given that the algorithm is coping with a dynamic data set. The degraded purity during the initial clustering results was found to be due to the presence of all seven canopy types during the initial portion of the data combined with a lack of prior evidence of cluster distributions. Examining the confusion matrix throughout the stream processing shows that this initial poor classification does not reoccur later with the concurrent reappearance of all seven classes.

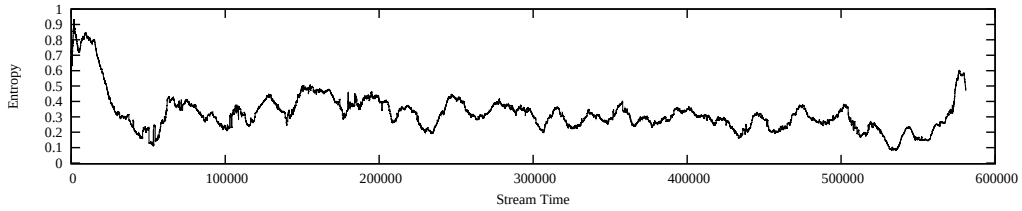
RepStream's purity measurements were evaluated against HPStream and CluStream using the results published in [5]. Figure 23 depicts the result of



(a)

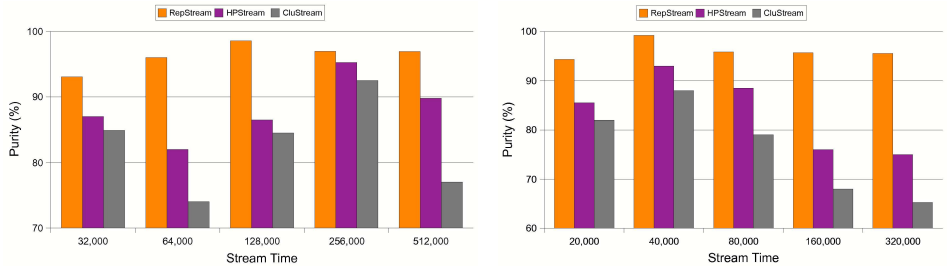


(b)



(c)

Fig. 22. Purity measure throughout the tree cover data stream over (a) a horizon of $h = 200$ points, (b) a horizon of $h = 1,000$ points and (c) the cluster entropy when $k = 5$ and $\alpha = 1.5$.



(a) (b)

Fig. 23. Comparative purity measures of RepStream, HPStream and CluStream using available published results on the forest tree cover data set for (a) $h = 200$ and (b) $h = 1,000$. RepStream parameters were $k = 5$ and $\alpha = 1.5$.

this comparison, showing that the algorithm was able to classify the tree data with consistently higher accuracy than the competing algorithms.

5.4 Parameter Sensitivity

The sensitivity of the algorithm with respect to the the neighbourhood connectivity and the density scaler parameters was explored using different values

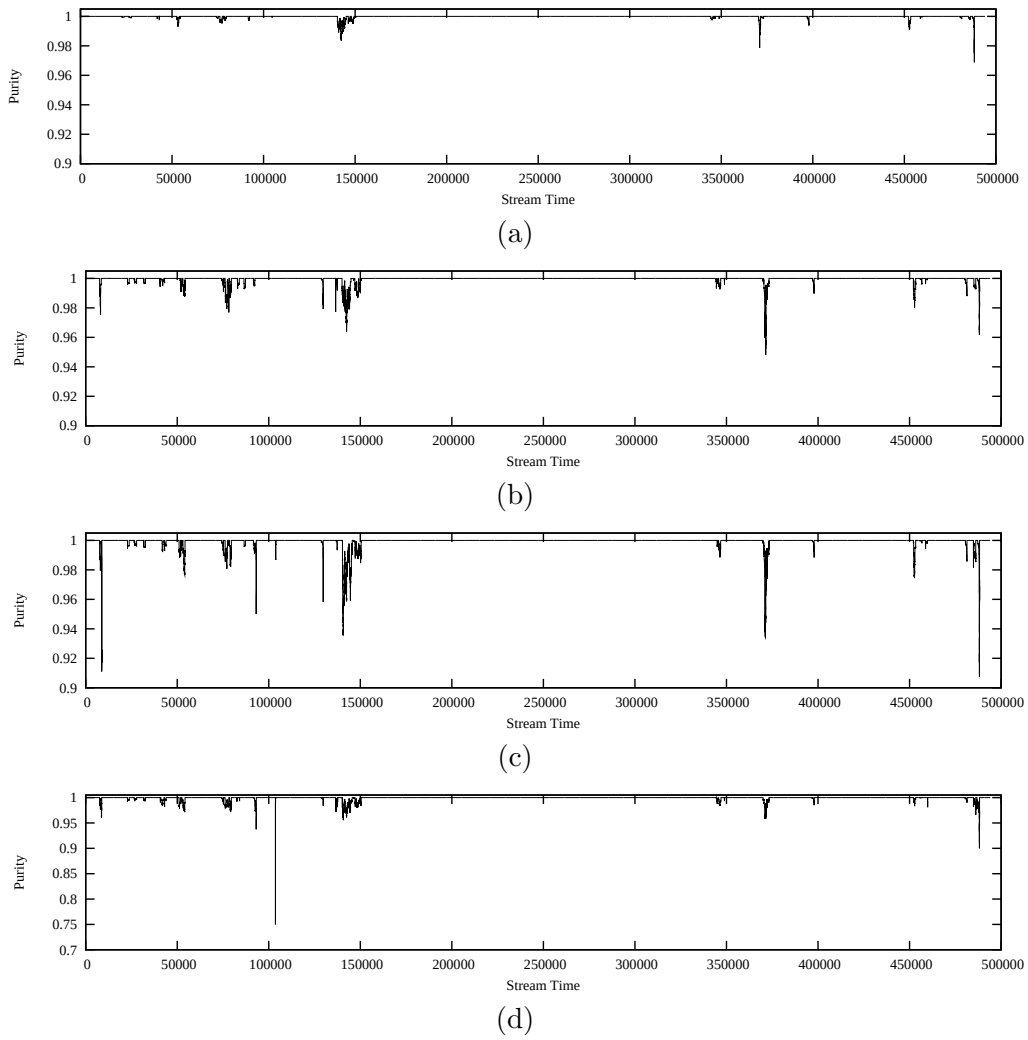


Fig. 24. Connectivity sensitivity: purity measure throughout the network intrusion data stream over a horizon of 1,000 points using density scaler $\alpha = 1.5$ with (a) $k = 3$, (b) $k = 7$, (c) $k = 9$ and (d) $k = 11$.

of k and α. A purity horizon of 1,000 points was used in all results shown.

Figure 24 depicts the purity measure of RepStream on the network intrusion data for $k = 3$, $k = 7$, $k = 9$ and $k = 11$ using a fixed scaling value of $\alpha = 1.5$. The results show that although the clustering performance of the algorithm degrades on this data set as k is increased, reasonable results are still produced up to $k = 9$ in Figure 24c. Comparing the results for $k = 7$ and $k = 9$ in Figure 24a and Figure 24b against those obtained for $k = 5$ in Figure 19b, we see that only minor differences exist between these connectivity values. The clusters discovered for the $k = 3$ parameter were, however, highly fragmented. This fragmentation was progressively reduced as the value of k was increased with large natural clusters being discovered with $k \geq 7$.

Similar results were obtained when the value of k was increased for the tree

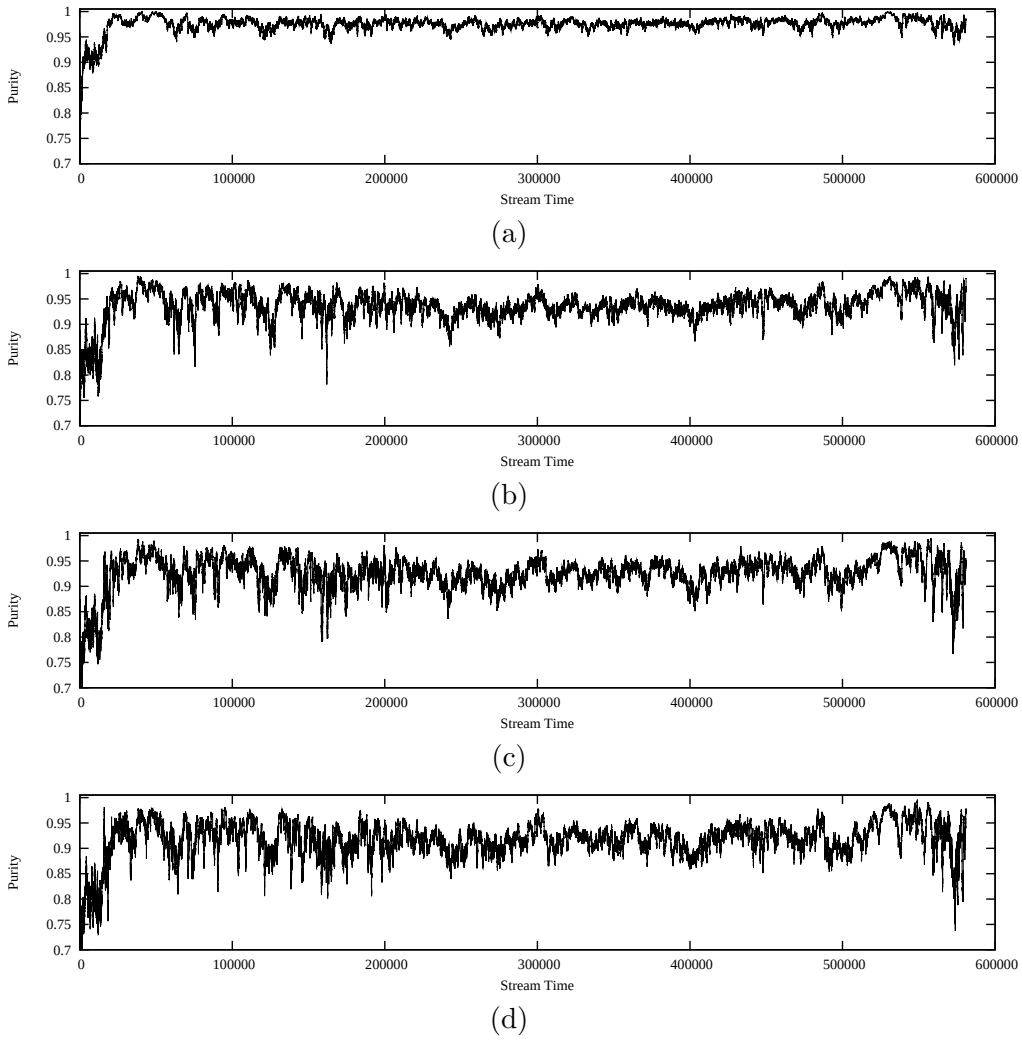


Fig. 25. Connectivity sensitivity: purity measure throughout the tree cover data stream over a horizon of 1,000 points using density scaler $\alpha = 1.5$ with (a) $k = 3$, (b) $k = 7$, (c) $k = 9$ and (d) $k = 11$.

cover data stream. The results, seen in Figure 25, again show a gradual increase in classification error as the neighbourhood connectivity is increased using values of $k = 3$, $k = 7$, $k = 9$ and $k = 11$ with a fixed density scaler of $\alpha = 1.5$. Results for $k = 5$ were given in Figure 22b. Fragmentation was, again, found to be an issue using $k = 3$ although this was resolved with values $k \geq 7$.

The magnitude of the performance decrease that has been observed in both the network intrusion and tree canopy data sets suggests that the clustering accuracy of RepStream degrades gracefully when sub-optimal values of k are chosen.

Algorithm sensitivity with regards to the density scaler was tested next. Results obtained on the network intrusion data set, seen in Figure 26, show, on

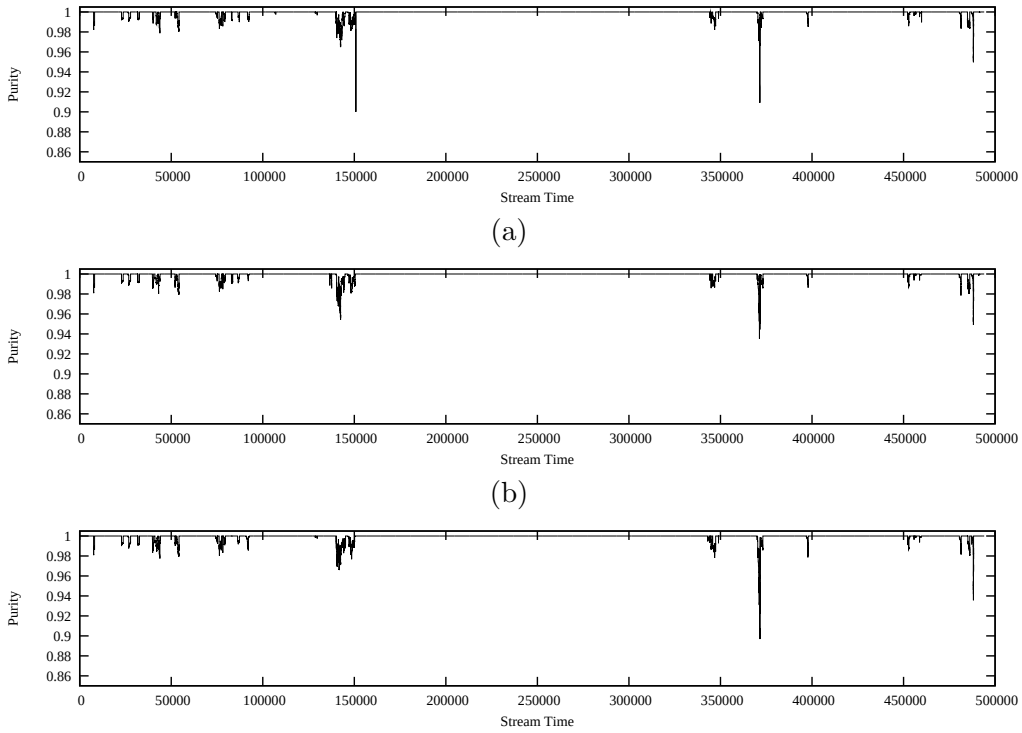


Fig. 26. Density scaler sensitivity: purity measure throughout the network intrusion data stream over a horizon of 1,000 points using connectivity $k = 5$ with (a) $\alpha = 3.0$, (b) $\alpha = 4.5$ and (c) $\alpha = 6.0$.

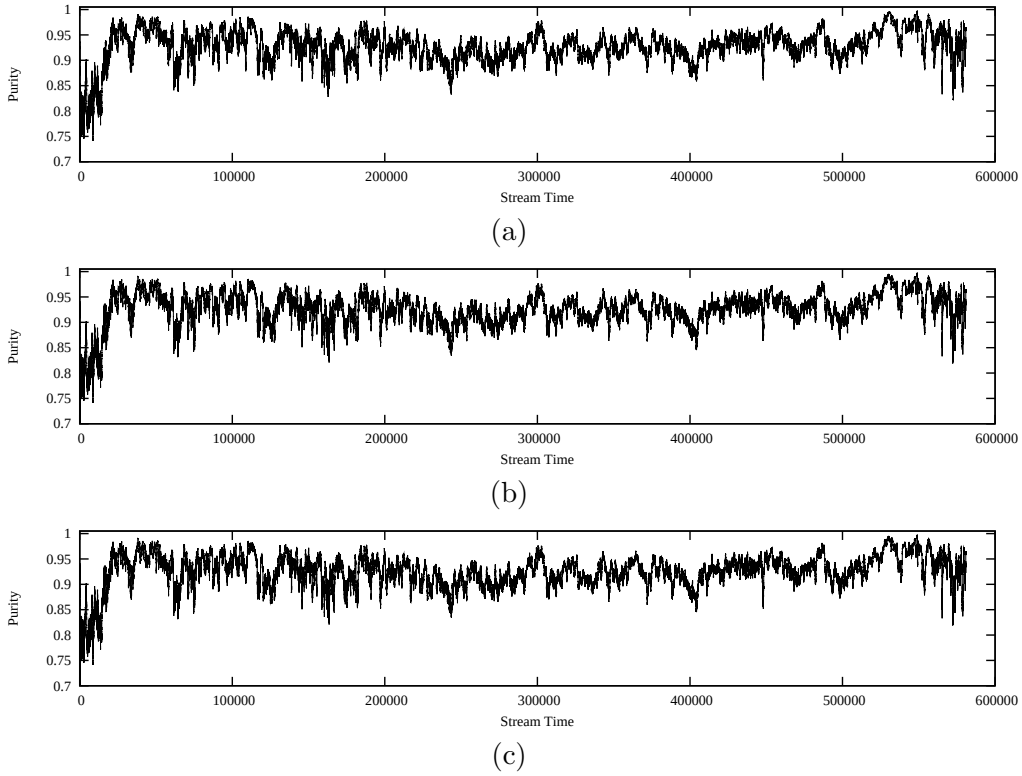
average, only marginal decreases in accuracy as the scaler value is increased using values of $\alpha = 3.0$, $\alpha = 4.5$ and $\alpha = 6.0$. However, contrasting this with $\alpha = 1.5$ in Figure 22b, we see that the worst case performance of the algorithm degrades significantly. Lower performance is particularly evident at times marking the beginning and end of some network attacks.

Differences observed on the tree cover data in Figure 27 was similarly minimal for values of $\alpha = 3.0$, $\alpha = 4.5$ and $\alpha = 6.0$. The choice of density scaler in this experiment has made no significant impact on the quality of the clustering. Results for $\alpha = 1.5$ were presented in Figure 22b.

Results shown on both the network intrusion and tree canopy data sets suggests that the performance of RepStream is relatively stable with respect to the choice of density scaler.

5.5 Scale-up Experiments

Several scale-up experiments were performed to investigate how well the execution time of the algorithm scales with respect to neighbourhood connectivity,



(a)

(b)

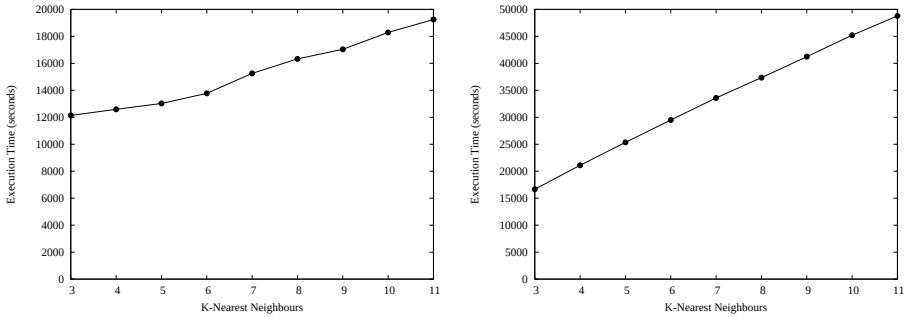
(c)

Fig. 27. Density scaler sensitivity: purity measure throughout the tree cover data stream over a horizon of 1,000 points using parameters $k = 5$ with (a) $\alpha = 3.0$, (b) $\alpha = 4.5$ and (c) $\alpha = 6.0$.

the density scaling factor, available memory and the length of the data stream. All experiments were run on an Intel 2.33 GHz Core 2 Duo processor under Mac OS 10.4. The RepStream implementation was single-threaded and used only a single processor core in each execution. Unless state otherwise, connectivity was set to $k = 5$ and a density scaler of $\alpha = 1.5$ was used to process both data sets in these experiments.

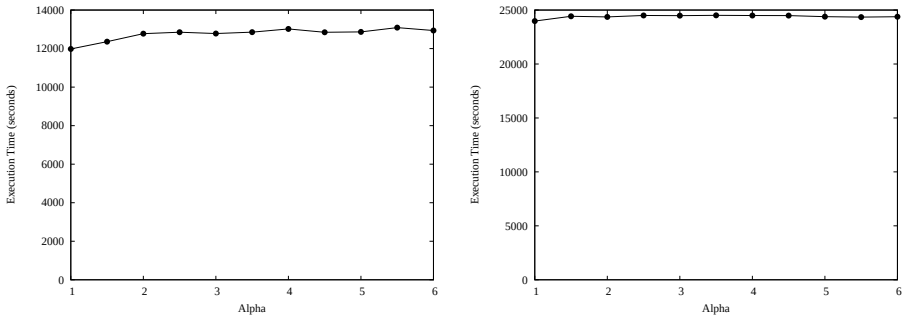
A near linear relationship between connectivity and execution time was discovered in the network intrusion results in Figure 28a. The forest data set produced a similar relationship as shown in Figure 28b. Scale-up results for increasing values of α are presented in Figure 29a and Figure 29b for the KDD and forest cover data sets, respectively. Both graphs show a slight increase in execution times as the scaling factor is increased from $\alpha = 1.0$ to $\alpha = 2.0$. The execution time with respect to the scaling factor is relatively linear for higher values of α. Near linear relationships with respect to execution time and available memory were also discovered and these are illustrated by the results shown in Figure 30.

Finally, the execution time of RepStream with respect to the length of the data stream is shown in Figure 31. An interesting contrast between the network intrusion and tree canopy data sets is noticeable here. Whereas the tree data



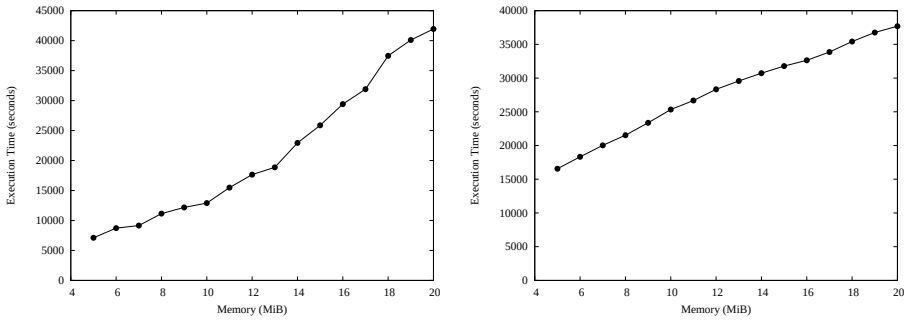
(a) (b)

Fig. 28. Execution time of RepStream clustering (a) the network intrusion data and (b) the forest canopy data as the k-nearest neighbours are increased. The scaling factor was set to $\alpha = 1.5$ in both experiments.



(a) (b)

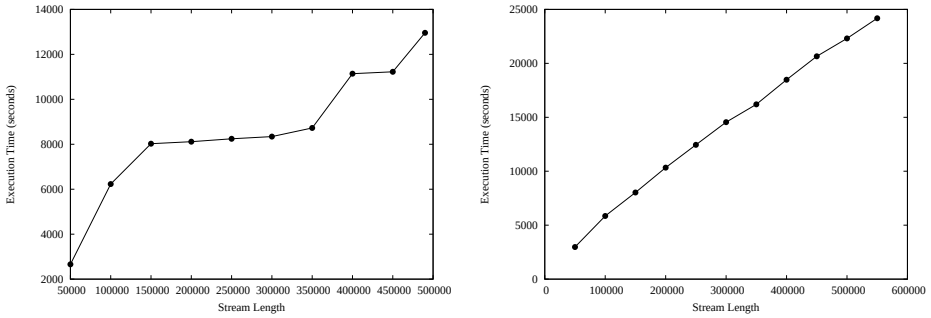
Fig. 29. Execution time of RepStream clustering (a) the network intrusion data and (b) the forest canopy data as the α scaling factor is increased. The connectivity was set to $k = 5$ in both experiments.



(a) (b)

Fig. 30. Execution time of RepStream clustering the (a) network intrusion data and (b) the forest canopy data as memory allocation is increased. The parameters used in both experiments were $k = 5$ and $\alpha = 1.5$.

set in Figure 31b shows an expected linear relationship between the number of points processed and the execution time, the network data set in Figure 31a displays a significant flattening out between stream time 150,000–350,000 and again between 400,000–450,000. This behaviour is a direct result of many identical points occurring within the stream at these times and the efficient



(a) (b)

Fig. 31. Execution time of RepStream clustering the (a) network intrusion data and (b) the forest canopy data with $k = 5$ and $\alpha = 1.5$ as the stream length is increased.

processing of them by singularity vertices.

6 Conclusions

This paper has introduced a graph-based incremental algorithm for clustering evolving stream data. We have selected a graph-based description because it allows us to model the spatio-temporal relationships in a data stream more accurately than it is possible using only summary statistics. Specifically, the graph allows a more detailed definition of the cluster boundary (rather than using a radius based measure which implies that the cluster has circular properties) and provides an effective and detailed representation of any changes that may occur in the cluster shape over time (the changes are reflected in the way in which the vertices and the connectivity changes over time). A key aspect of our is the use of representative points within the graph description allow the algorithm to capture the general structure of the clusters without requiring the complete cluster data to be stored in memory. The most pertinent of the representative vertices are stored inside a repository which is used to recall previously discovered cluster structures in the presence of recurrent change.

The algorithm was shown to prioritise the retention of important information within the repository by weighting the usefulness of the representative vertices with respect to time. This enables the algorithm to discard the least useful cluster features when memory constraints mandate that some data be discarded. The repository can also be used to obtain a historical perspective on the general shape and distribution of the clusters over time and to archive these changes for off-line analysis.

Experimental results demonstrated that the algorithm is able to effectively classify both synthetic and real world data sets. The algorithm was compared

against an incremental implementation of DBSCAN and shown to robustly handle clusters of complex shapes, sizes and densities. DBSCAN, in comparison, is shown to be hampered by a static density threshold ill suited towards stream processing. Results on real world data sets showed that RepStream was able to more accurately classify well known network intrusion and forest canopy data sets than three of the most popular stream data clustering algorithms: DenStream, HPStream and CluStream.

An integral component of the algorithm is the knowledge repository, a collection of the most useful representative vertices that define the shape and distribution characteristics of important cluster structures. The repository was shown to assist with the clustering of real world network intrusion data by retaining historical cluster features that aid classification of data characteristic of recurrent change.

Investigation into parameter sensitivity revealed that RepStream is fairly resistant to sub-optimal selections of the nearest neighbour connectivity parameter k and highly resistant to changes in the density scaler α. Conservative values for either parameter were observed to result in the discovery of fragmented clusters whereas only higher values of k tended to introduce classification errors due to incorrect cluster merges.

Acknowledgements

The authors wish to thank Dinh Q. Phung for the DBSCAN implementation used in this paper.

References

[1] C. C. Aggarwal, Re-designing distance functions and distance-based applications for high dimensional data, ACM SIGMOD Record 30 (1) (2001) 13–18.

[2] C. C. Aggarwal, Towards systematic design of distance functions for data mining applications, in: Proc. 9th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, ACM Press, Washington, District of Columbia, USA, 2003.

[3] C. C. Aggarwal, A human-computer interactive method for projected clustering, IEEE Trans. Knowledge and Data Engineering 16 (4) (2004) 448–460.

[4] C. C. Aggarwal, J. Han, J. Wang, P. Yu, A framework for clustering evolving data streams, in: Proc. 29th Int'l Conf. Very Large Data Bases, Berlin, Germay, 2003.

[5] C. C. Aggarwal, J. Han, J. Wang, P. S. Yu, A framework for projected clustering of high dimensional data streams, in: M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, K. B. Schiefer (eds.), Proc. 30th Int'l Conf. Very Large Data Bases, Morgan Kaufmann, Toronto, Canada, 2004.

[6] C. C. Aggarwal, J. Han, P. S. Yu, On demand classification of data streams, in: Proc. 10th ACM SIGKDD Int'l Conf. Knowledge discovery and data mining, ACM Press, Seattle, Washington, USA, 2004.

[7] C. C. Aggarwal, P. S. Yu, A framework for clustering uncertain data streams, in: IEEE 24th Int'l Conf. Data Engineering, Cancún, México, 2008.

[8] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghaven, Automatic subspace clustering of high dimensional data for data mining applications, ACM SIGMOD Record 27 (2) (1998) 94–105.

[9] M. Ankerst, M. M. Breunig, H.-P. Kriegel, J. Sander, OPTICS: Ordering points to identify the clustering structure, in: A. Delis, C. Faloutsos, S. Ghandeharizadeh (eds.), Proc. ACM SIGMOD Int'l Conf. Management of Data, ACM Press, Montreal, Canada, 1999.

[10] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, J. Widom, Data-Stream Management: Processing High-Speed Data Streams, chap. STREAM: The Stanford Data Stream Management System, Springer-Verlag, 2005.

[11] A. Asuncion, D. J. Newman, UCI machine learning repository (2007).
URL `http://mlearn.ics.uci.edu/MLRepository.html`

[12] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proc. 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM Press, Madison, Wisconsin, USA, 2002.

[13] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, S. Datta, Graph clustering and minimum cut trees, Information Sciences 176 (14) (2006) 1952–1985.

[14] J. L. Bentley, Mutlidimensional binary search trees used for associative searching, Communications of the ACM 18 (9) (1975) 509–517.

[15] J. Beringer, E. Hüllermeier, Online clustering of parallel data streams, Data & Knowledge Engineering 58 (2) (2006) 180–204.

[16] J. A. Blackard, D. J. Dean, Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables, Computers and Electronics in Agriculture 24 (3) (1999) 131–151.

[17] Y. D. Cai, D. Clutter, G. Pape, J. Han, M. Welge, L. Auvil, MAIDS: Mining alarming incidents from data streams, in: Proc. ACM SIGMOD Int'l Conf. Management of Data, Paris, France, 2003.

[18] F. Cao, M. Ester, W. Qian, A. Zhou, Density-based clustering over an evolving data stream with noise, in: J. Ghosh, D. Lambert, D. B. Skillicorn, J. Srivastava (eds.), Proc. Sixth SIAM Int'l Conf. Data Mining, SIAM, Bethesda, Maryland, USA, 2006.

[19] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. Zdonik, Monitoring streams: A new class of data management applications, in: Proc. 28th Int'l Conf. Very Large Data Bases, 2002.

[20] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, S. B. Zdonik, Scalable distributed stream processing, in: Proc. First Biennia Conf. Innovative Data Systems Research, 2003.

[21] J. Choo, R. Jiamthapthaksin, C.-S. Chen, O. U. Celepcikay, C. Giusti, C. F. Eick, MOSAIC: A proximity graph approach for agglomerative clustering, in: I. Y. Song, J. Eder, T. M. Nguyen (eds.), 9th Int'l Conf. Data Warehousing and Knowledge Discovery, vol. 4654 of Lecture Notes in Computer Science, Springer, Regensburg, Germany, 2007.

[22] P. Domingos, G. Hulten, Catching up with the data: Research issues in mining data streams, in: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Santa Barbara, California, USA, 2001.

[23] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, X. Xu, Incremental clustering for mining in a data warehousing environment, in: A. Gupta, O. Shmueli, J. Widom (eds.), Proc. 24rd Int'l Conf. Very Large Data Bases, Morgan Kaufmann, New York City, New York, USA, 1998.

[24] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: E. Simoudis, J. Han, U. M. Fayyad (eds.), Proc. 2nd Int'l Conf. Knowledge Discovery and Data Mining, AAAI Press, Portland, Oregon, USA, 1996.

[25] G. W. Flake, R. E. Tarjan, K. Tsioutsiouliklis, Graph clustering and minimum cut trees, Internet Mathematics 1 (4) (2003) 385–408.

[26] M. M. Gaber, A. B. Zaslavsky, S. Krishnaswamy, Mining data streams: A review, SIGMOD Record 34 (2) (2005) 18–26.

[27] V. Ganti, J. Gehrke, R. Ramakrishnan, DEMON: Mining and monitoring evolving data, IEEE Trans. Knowledge and Data Engineering 13 (1) (2001) 50–63.

[28] J. Gao, J. Li, Z. Zhang, P.-N. Tan, An incremental data stream clustering algorithm based on dense units detection, in: T. B. Ho, D. W.-L. Cheung, H. Liu (eds.), Proc. 9th Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining, vol. 3518 of Lecture Notes in Computer Science, Springer, 2005.

[29] S. Guha, A. Meyerson, N. Mishra, R. Motwani, L. O'Callaghan, Clustering data streams: Theory and practice, IEEE Trans. Knowledge and Data Engineering 15 (3) (2003) 515–528.

[30] L. Hagen, A. B. Kahng, A new approach to effective circuit clustering, in: Proc. IEEE/ACM Int'l Conf. Computer-Aided Design, IEEE Computer Society, Santa Clara, California, USA, 1992.

[31] D. Harel, Y. Koren, On clustering using random walks, in: R. Hariharan, M. Mukund, V. Vinay (eds.), Proc. 21st Conf. Foundations of Software Technology and Theoretical Computer Science, vol. 2245 of Lecture Notes in Computer Science, Springer, 2001.

[32] J. A. Hartigan, Clustering Algorithms, John Wiley & Sons, New York City, New York, USA, 1975.

[33] J. He, A.-H. Tan, C. L. Tan, S. Y. Sung, On quantitative evaluation of clustering systems, in: W. Wu, H. Xiong, S. Shekhar (eds.), Clustering and Information Retrieval, Kluwer, 2003, pp. 105–134.

[34] S. Hettich, S. D. Bay, The UCI KDD archive (1999).
URL `http://kdd.ics.uci.edu`

[35] A. Hinneburg, C. C. Aggarwal, D. A. Keim, What is the nearest neighbor in high dimensional spaces?, in: A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, K.-Y. Whang (eds.), Proc. 26th Int'l Conf. Very Large Data Bases, Morgan Kaufmann, Cairo, Egypt, 2000.

[36] A. K. Jain, R. C. Dubes, Algorithms for Clustering Data, Prentice-Hall, 1988.

[37] A. K. Jain, M. N. Murty, P. J. Flynn, Data clustering: A review, ACM Computing Surveys 31 (3) (1999) 264–323.

[38] R. Kannan, S. Vempala, A. Vetta, On clusterings: Good, bad and spectral, in: Proc. 41st Annual Symposium Foundations of Computer Science, IEEE Computer Society, Redondo Beach, California, USA, 2000.

[39] R. Kannan, S. Vempala, A. Vetta, On clusterings: Good, bad and spectral, Journal of the ACM 51 (3) (2004) 497–515.

[40] G. Karypis, R. Aggarwal, V. Kumar, S. Shekhar, Mutlilevel hypergraph partitioning: Application in VLSI domain, IEEE Trans. Very Large Scale Integration (VLSI) Systems 7 (1) (1999) 69–79.

[41] G. Karypis, E.-H. Han, V. Kumar, Chameleon: Hierarchical clustering using dynamic modeling, Computer 32 (8) (1999) 68–75.

[42] D. Knuth, The Art of Computer Programming, vol. 3, 3rd ed., Addison-Wesley, 1997.

[43] S. Lühr, M. Lazarescu, Connectivity based stream clustering using localised density exemplars, in: Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining, vol. 5012 of Lecture Notes in Computer Science, Osaka, Japan, 2008.

[44] J. B. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, University of California, 1967.

[45] A. W. Moore, Efficient memory-based learning for robot control, Ph.D. thesis, Cambridge University (1990).

[46] O. Nasraoui, C. Rojas, C. Cardona, A framework for mining evolving trends in web streams using dynamic learning and retrospective validation, Computer Networks 50 (10) (2006) 1488–1512.

[47] O. Nasraoui, C. C. Uribe, C. R. Coronel, F. Gonzalez, TECNO-STREAMS: Tracking evolving clusters in noisy data streams with a scalable immune system learning model, in: IEEE Int'l Conf. Data Mining, IEEE Computer Society, Washington, District of Columbia, USA, 2003.

[48] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, Streaming-data algorithms for high-quality clustering, in: Proc. 18th Int'l Conf. Data Engineering, IEEE Computer Society, Washington, District of Columbia, USA, 2002.

[49] N. H. Park, W. S. Lee, Cell trees: An adaptive synopsis structure for clustering multi-dimensional on-line data streams, Data & Knowledge Engineering 63 (2) (2007) 528–549.

[50] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, P. K. Chan, Cost-based modeling for fraud and intrusion detection: Results from the JAM project, in: Proc. DARPA Information Survivability Conference and Exposition, vol. 2, Hilton Head, South Carolina, USA, 2000.

[51] D. K. Tasoulis, G. Ross, N. M. Adams, Visualising the cluster structure of data streams, in: M. R. Berthold, J. Shawe-Taylor, N. Lavrac (eds.), Proc. 7th Int'l Symposium on Intelligent Data Analysis, vol. 4723 of Lecture Notes in Computer Science, Springer, 2007.

[52] S. van Dongen, Graph clustering by flow simulation, Ph.D. thesis, University of Utrecht (May 2000).

[53] R. Xu, D. Wunsch II, Survey of clustering algorithms, IEEE Trans. Neural Networks 16 (3) (2005) 645–678.

[54] T. Zhang, R. Ramakrishnan, M. Livny, BIRCH: An efficient data clustering method for very large databases, in: H. V. Jagadish, I. S. Mumick (eds.), Proc. ACM SIGMOD Int'l Conf. Management of Data, ACM Press, 1996.

[55] A. Zhou, F. Cao, Y. Yan, C. Sha, X. He, Distributed data stream clustering: A fast EM-based approach, in: Proc. 23rd Int'l Conf. Data Engineering, IEEE, 2007.

[56] Y. Zhou, K. Aberer, A. Salehi, K.-L. Tan, Rethinking the design of distributed stream processing systems, in: Proc. 24th Int'l Conf. Data Engineering Workshops, ICDE 2008, Cancún, México, 2008.