# Summarizing Cluster Evolution in Dynamic Environments

Irene Ntoutsi[1,3], Myra Spiliopoulou[2], Yannis Theodoridis[3]

[1]Institute for Informatics, Ludwig-Maximilians-Universität München, Germany
http://www.dbs.ifi.lmu.de
ntoutsi@dbs.ifi.lmu.de
[2]School of Computer Science, University of Magdeburg, Magdeburg, Germany
http://omen.cs.uni-magdeburg.de/itikmd/home/index.html
myra@iti.cs.uni-magdeburg.de
[3]Department of Informatics, University of Piraeus, Greece
http://infolab.cs.unipi.gr/
ytheod@unipi.gr

**Abstract.** Monitoring and interpretation of changing patterns is a task of paramount importance for data mining applications in dynamic environments. While there is much research in adapting patterns in the presence of drift or shift, there is less research on how to maintain an overview of pattern changes over time. A major challenge lays in summarizing changes in an effective way, so that the nature of change can be understood by the user, while the demand on resources remains low. To this end, we propose FINGERPRINT, an environment for the summarization of cluster evolution. Cluster changes are captured into an "evolution graph", which is then summarized based on cluster similarity into a *fingerprint of evolution* by merging similar clusters. We propose a batch summarization method that traverses and summarizes the Evolution Graph as a whole, and an incremental method that is applied during the process of cluster transition discovery. We present experiments on different data streams and discuss the space reduction and information preservation achieved by the two methods.

## 1 Introduction

Data streams are used in many modern applications and impose new challenges for the data management systems because of their size and high degree of variability. One of the challenges is the efficient detection and monitoring of changes in the underlying population. For example, changes in the patterns known to a network intrusion detection system may indicate that intruders test new attacks and abandon old, already known (and blocked) intrusion patterns. In general, monitoring of change is essential for applications demanding long–term prediction and proaction. While much research has been recently devoted to pattern change detection, little work has been done on the efficient maintenance of the pattern changes.

The maintenance and summarization of pattern changes upon a stream is a new problem. Summarization of data (rather than patterns), however, has been studied extensively: Popular summarization methods include histograms and wavelets, and there is much work on the efficient maintenance of these structures and on the adaptation of their contents when data change; however, these methods do not show how the data change nor do they maintain the changes themselves. There is also research on storing, modifying and querying patterns in inductive or conventional databases (e.g. [4]); however, those approaches have not been designed for patterns over streams and, although there is provision for modifying patterns when new data arrive, there are no solutions on the efficient maintenance of changes over time. Finally, there are methods for pattern change detection (e.g. [1, 12, 13]), in which different types of change can be identified and highlighted; however, the efficient long-term maintenance of the changes over an "infinite" stream is not considered.

Evolution is a permanent characteristic of streamed data, thus long-term perusal requires a space-efficient accommodation of the evolving patterns and a representation that highlights remarkable changes while suppressing trivial pattern perturbations. In this study, we propose a graph representation of pattern changes/transitions and two algorithms that condense this graph into a "fingerprint" - a structure in which similar patterns are efficiently summarized, subject to an information loss function.

The rest of the paper is organized as follows: Related work is discussed in Section 2. In Section 3, we present our graph model for the representation of cluster transitions. The criteria for the summarization of cluster changes and the actual summarization methods are presented in Section 4. Experiments are presented in Section 5. Section 6 concludes our work.

## 2 Related Work

*Summarization methods:* Summarization for a set of transactions with categorical attributes is studied by Chandola and Kumar [6]. In one of their methods, they derive summaries by clustering the transactions, extracting the feature/value pairs supported by all transactions in a cluster and using them as cluster summaries. They do not address the issue of cluster change upon a stream, but propose two metrics that characterize the output of the summarization algorithm, "compaction gain" and "information loss". Quite naturally, our metrics are similarly motivated and have similar names. However, they summarize static data using clusters, while we summarize evolving clusters upon an "infinite" stream.

Summarization and change are considered by [10], who study changes of database content summaries. They define as "content summary" for a database a set of keywords, weighted on their importance within the database. Meta-search services use such summaries to select appropriate databases, towards which they issue keyword-based queries. The reliability of such a summary deteriorates as the contents of the database change over time. So, the authors propose methods to quantify and detect summary changes. This study addresses both the issue of summarization over the evolving database and the discovery of changes. However, the maintenance of the summaries themselves in a condensed form is beyond the scope of their work. On the other hand, the proposed FINGERPRINT method emphasizes on the summarization of the discovered population transitions.

The discovery and representation of cluster changes for text streams are studied by [12]. They apply soft clustering with mixture models at each time period, extract the representative keyword-list ("theme") for each cluster and then monitor the evolution of these lists by tracing divergences between a current keyword list and past ones. Theme transitions are maintained on a "theme evolution graph", which is then used to extract the life cycle of themes (through Hidden Markov Models). The graph structure is used to reflect pattern changes, but the maintenance of this evergrowing graph is not studied and the need for summarizing it without losing information is not anticipated.

*Stream clustering:* Relevant to our work is the work on stream clustering. Usually, storing an entire data stream or scanning a stream multiple times is impossible due to its tremendous volume [9]. To this end, several clustering algorithms have been proposed which aggregate the stream online through some appropriate summary structure and cluster these summaries offline. This rationale was first introduced in CluStream [2]; the summary structure, called micro-cluster, is a temporal extension of the cluster feature vector of BIRCH [15]. CluStream starts with $k$ initial micro–cluster summaries and as new points arrive, the summaries are updated such that a total of $k$ micro-clusters is maintained at each time point. The clusters are detected offline using a modified version of $k$-Means over summaries instead of raw data; the user chooses the summaries to be considered by specifying the time interval. Micro-clusters can be observed as cluster summaries and are indeed designed to reduce space demand. Nonetheless, CluStream focuses on combining them into clusters rather than in summarizing them. Also, the information loss effected through summarization is not discussed. The same holds for DENstream [5] and DStream [7], which also follow the online–offline rationale.

*Change detection:* Change detection methods are also relevant to our work. Aggarwal [1] models clusters through kernel functions and changes as kernel density changes at each spatial location of the trajectory. The emphasis is on computing change velocity and finding the locations with the highest velocity - the epicenters. This model of change is very powerful, but is restricted to data over a fixed feature space. Kalnis et al [11] propose a special type of cluster change, the moving cluster, whose contents may change while its density function remains the same during its lifetime. They find moving clusters by tracing common data records (based on their IDs) between clusters of consecutive timepoints. Yang et al [14] detect formation and dissipation events upon clusters of spatial scientific data. Their framework supports four types of spatial object association patterns (SOAP), namely Star, Clique, Sequence, and minLink, which are used to model different interactions among spatial objects. Such methods however, assume that the feature space does not change. Thus, they cannot be used for dynamic feature spaces, e.g.n text stream mining, where features are usually frequent words. Furthermore, hierarchical clustering algorithms cannot be coupled with such a method. Cluster transition modeling and detection methods are presented in the MONIC framework of [13], where both changes in the data and in the feature space are anticipated. Differently from the model of [1], MONIC covers changes that involve more than one cluster (*external transitions*), such as split and absorption, allowing insights in the whole clustering. *Internal transitions*, i.e. changes within a single cluster (shrink, shift etc.), are also supported. The transition tracking mechanism of MONIC is based on the contents of the underlying data stream, thus it is independent of the clustering algorithm and of the cluster summarization method (differently from [12]). For these reasons, we use the cluster transition model of MONIC as input to our methods for the summarization of cluster changes.

## 3 Building the Evolution Graph

We model cluster evolution across a sequence of timepoints $t_1, \ldots, t_n$ and denote as $\xi_1, \ldots, \xi_n$ the clusterings discovered at those timepoints. A clustering $\xi_i, i > 1$ may be the result of a complete re-clustering at $t_i$ or of the adaptation of clustering $\xi_{i-1}$. We further denote as $d_i$ the substream of data records seen in the interval $(t_{i-1}, t_i]$ and as $D_i$ the substream of records, on which $\xi_i$ is based. Depending on whether data ageing is considered or not, $D_i$ may be equal to the set of all records seen thus far ($D_i = \cup_{j=1}^{i} d_j$) or to the substream seen within a time window.

The *Evolution Graph* $EG \equiv G(V, E)$ spans the whole period of observation ($n$ timepoints). The set of nodes $V$ corresponds to the set of clusters seen during this time period, i.e. $V = \{\xi_1, \ldots, \xi_n\}$. The set of edges $E$ contains the cluster transitions; For each $e = (v, v') \in E$, there is a timepoint $t_i, 1 \leq i < n$ such that $v \in \xi, v' \in \xi_{i+1}$. By this specification of the Evolution Graph, the edges connect nodes/clusters found at adjacent timepoints. An example is depicted in Fig. 1: A *dotted/green* edge denotes a "split" of the source cluster to multiple target clusters. A *dashed/orange* edge describes an "absorption"; the source cluster is contained in the target cluster. A *solid/blue* edge indicates a "survival"; the source cluster has survived into the target cluster with minor changes, such as changes in size or homogeneity [13]. This example graph depicts three

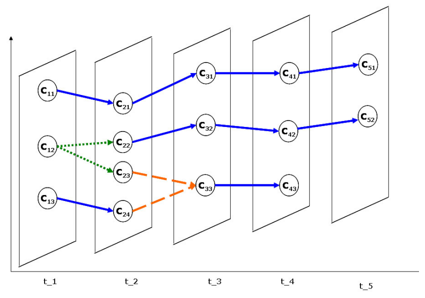

**Fig. 1.** Example of an Evolution Graph ($EG$)

types of cluster transitions: split, absorption and survival [13]. In the next subsection, we describe how we assign the semantics of those transitions to the edges of the graph.

### 3.1 Semantics of the Graph Nodes

A node $c \in V$ represents a cluster found at timepoint $t_i$, i.e. belonging to clustering $\xi_i$. A node in the evolution graph is adorned with a "label" $c.label$ or $\widehat{c}$, i.e. an intensional/

summarized representation of its members. There are many elaborate summarized representations proposed in the literature, including micro-clusters [2] and "droplets" over text data [3]. We opt for two simple representations, the cluster's centroid for clusters over arbitrary numerical data and the cluster's topic for clusters over text data.

**Definition 1 (Centroid as Label).** *Let $c$ be a cluster in an $m$-dimensional space of numerical properties. Its centroid is the vector of the mean values $c.label \equiv \widehat{c} :=< \mu_1 \ldots \mu_m >$, where $\mu_l$ is the average of the data records' values across the $l^{th}$-dimension.*

**Definition 2 (Keyword-based label).** *Let $c$ be a cluster of text documents, where each document $d_i \in c$ is a vector in the feature space of the keywords $\{k_1, \ldots, k_m\}$. The cluster label is defined as $c.label \equiv \widehat{c} :=< w_{k_1}, \ldots, w_{k_m} >$, where $w_{k_l}$ is (a) the frequency of the $l^{th}$-keyword within $c$, if this frequency exceeds a boundary $b$ and (b) zero otherwise.*

## 3.2 Semantics of the Graph Edges

An edge $e = (c, c') \in E$ denotes that a cluster $c \in \xi_i$ found at $t_i$ has been "succeeded" by a cluster $c' \in \xi_{i+1}$ of the next timepoint. Succession means that among the clusters of $\xi_{i+1}$, the cluster $c'$ is the one most similar to the cluster $c$. The semantics of cluster succession can be designed according to any of the approaches proposed for cluster evolution monitoring (e.g. [1, 12, 13]). We have opted for the MONIC approach [13] because it is independent of the the clustering algorithm and can thus be used for any type of clusters (in contrary to e.g. [1]). Also, it considers ageing of data which is important for streams. In MONIC, cluster succession is based on the notions of *cluster overlap* and *cluster matching*: Let $c$ be a cluster in clustering $\xi_i$ at $t_i$ and $c'$ be a cluster in clustering $\xi_{i+1}$ at $t_{i+1}$. The overlap of $c$ and $c'$ is defined as: $overlap(c, c') = \frac{|c \cap c'|}{|c|}$. This means that the overlap depends on the members of $c$ that are still remembered at $t_{i+1}$. Then, the "best match" or simply "match" of $c$ in $\xi_{i+1}$ is the cluster $c' \in \xi_{i+1}$ that has the maximum overlap to $c$, subject to a threshold $\tau_{match}$. If the threshold is not reached, then there is no match for $c$ in $\xi_{i+1}$, i.e. $c$ has disappeared/died.

The transitions that a cluster might encompass are:

1. *survival*, denoted as $c \rightarrow c'$: $c \in \xi_i$ survives into $c' \in \xi_{i+1}$ iff $c'$ is the match for $c$ and there is no other cluster $z \in \xi_i$, for which $c'$ is the match.
2. *absorption*, denoted as $c \overset{\subsetneq}{\rightarrow} c'$: $c \in \xi_i$ is absorbed by $c' \in \xi_{i+1}$ iff $c'$ is the match for $c$ and there is at least one more cluster $z \in \xi_i$, for which $c'$ is the match.
3. *split*, denoted as $c \overset{\supsetneq}{\rightarrow} \{c_1, \ldots, c_p\}$: $c \in \xi_i$ is split into $c_1, \ldots, c_p \in \xi_{i+1}$, with $p > 1$, iff the overlap of $c$ to each of these clusters exceeds a threshold $\tau_{split}$ and the overlap of all these clusters to together exceeds the match threshold $\tau$.
4. *disappearance*, denoted as $c \rightarrow \odot$: $c \in \xi_i$ disappears if none of the above mentioned cases holds for $\xi_{i+1}$.

In our Evolution Graph, an edge is drawn from $c$ to $c'$ for each of the first three cases; if a cluster has no outgoing edges, then the forth case has occurred. Further, we adorn the edges with information on the transition type. In particular, let $e = (c, c') \in E$

be an edge from cluster $c \in \xi_i$ to $c' \in \xi_{i+1}$. Then, the edge $e$ is adorned with a label $e.extTrans$ that describes the type of external transition as one of $\{survival, split, absorption\}$. If a cluster in $\xi_i$ has no outgoing edge, it has disappeared. If a cluster in $\xi_{i+1}$ has no ingoing edge, it has just *emerged*. For an emerged cluster, we form its *cluster trace*:

**Definition 3 (Cluster Trace).** *Let $EG$ be an Evolution Graph captured for the timepoints $t_1, \ldots, t_n$. For each emerged cluster c that appeared for the first time at $t_i$ (i.e. a cluster without ingoing edge), we define its "cluster trace", $trace(c) \equiv trace(c, t_i)$, as the sequence $\prec c_1 \cdot c_2 \cdot \cdot c_m \succ$, where $c_1 \equiv c$, $m \leq n - i$ and for each $c_i, i \geq 2$ there is an edge $e_i = (c_{i-1}, c_i)$ such that $e.extTrans = survival$. We denote the traceset of $EG$ as $\mathcal{T}_{EG}$.*

For the Evolution Graph of Fig. 1, the traceset $\mathcal{T}_{EG}$ consists of the following sequences: (a) trace $\prec c_{11}c_{21}c_{31}c_{41}c_{51} \succ$, indicating that the emerged cluster $c_{11}$ has survived across all five timepoints, (b) trace $\prec c_{22}c_{32}c_{42}c_{52} \succ$ of the emerged cluster $c_{22}$, one of the clusters to which $c_{12}$ has been split and (c) the two-node traces $\prec c_{13}c_{24} \succ$ and $\prec c_{33}c_{43} \succ$. The other clusters $c_{12}, c_{23}, c_{24}$ only existed for one timepoint and therefore built no traces.

### 3.3 Evolution Graph construction

The Evolution Graph is built incrementally as new clusterings arrive at $t_1, \ldots, t_n$. The pseudocode of the algorithm is depicted in Fig. 2. When a new clustering $\xi_i$ arrives at $t_i, i > 1$, MONIC [13] is applied on the previous clustering $\xi_{i-1}$ and the current one $\xi_i$ (line 4): transitions between clusters of $\xi_{i-1}, \xi_i$ are detected and an edge is added to the Evolution Graph for each detected transition adorned with information on the transition type (line 5). Clusters at $\xi_i$ are also added as nodes to the graph and labels are assigned to them (line 2). Note that MONIC uses the cluster contents (data members) for transition detection. Hence, we retain this information until the next timepoint only (line 6). So, the data members of the clusters at $t_{i-1}$ are retained only until $t_i$, so as the transitions between clusters at $t_i, t_{i-1}$ to be detected.

## 4 Summarizing the Evolution Graph

The Evolution Graph captures the whole history of the population under observation and allows the study of cluster transitions and the inspection of cluster interrelationships. However, this graph is space consuming and redundant. Concretely, it contains information about each change but also contains information for clusters that did not change at all. Hence, we summarize the Evolution Graph in such a way that cluster transitions are reflected but redundancies are omitted. For this, we summarize traces, i.e. sequences of cluster survivals, into "fingerprints". These trace summaries constitute the "fingerprint" of the Evolution Graph.

### 4.1 Summarizing a Trace

The summarization process is applied over cluster traces (c.f. Definition 3). Each trace $T$ is traversed and the "removable" nodes are identified: These are the nodes that can be

```
**BuildEG()**
Output: $EG = G(V, E)$ -- the Evolution Graph
begin
1.   while a new clustering $\xi\_i$ arrives at $t\_i$ begin
2.       $EG.addNodes(\xi\_i.nodes)$; //Add $\xi\_i$ clusters in $EG$
3.       if (i==1) then return $EG$; else $j = i - 1$; // just for notation
4.       $E\_ji = MONICtransitions(\xi\_j, \xi\_i)$; //Detect transitions
5.       $EG.addEdges(E\_ji)$; //Add transitions in $EG$
6.       $EG.updateNodes(\xi\_j.nodes)$;//Remove redundant information from $\xi\_j$
7.   end;
8.   return $EG$;
end
```

**Fig. 2.** The evolution graph (EG) construction algorithm

replaced by a smaller number of derived nodes, which are called "virtual centers" and are defined below.

**Definition 4 (Virtual Center).** *Let $\prec c_1 \ldots c_m \succ$ be the trace of an emerged cluster c, $trace(c)$ and let $X = \prec c_j \ldots c_{j+k} \succ$ be a subtrace of this trace, i.e. a subsequence of adjacent nodes in the trace ($k \leq m-1$, $j \geq 1$). We define the "virtual center" of $X$, $vcenter(X) \equiv \widehat{X}$ as a derived node composed of the averages of the labels of the nodes in $X$:*

$$\widehat{X}[i] = \frac{1}{|X|} \sum_{c_i \in X} \widehat{c}[i]$$

*where $\cdot[i]$ is the $i^{th}$ dimension and $\widehat{c}$ denotes the label of cluster c. We use the notation $c \mapsto \widehat{X}$ to indicate that cluster $c \in X$ has been "mapped to" the virtual center $\widehat{X}$.*

If labels are centroid-based (Definition 1), $\widehat{X}$ is the center of the centroids of the clusters in $X$. If labels are keyword-based (Definition 2), $\widehat{X}$ contains the average frequencies of all frequent keywords in the clusters of $X$.

After introducing the virtual center as the summary of a subtrace, we define the summary of a trace: It consists of a sequence of nodes, each node being either an original cluster or a virtual center that summarizes a subtrace.

**Definition 5 (Trace Summary).** *Let $T = \prec c_1 \ldots c_m \succ$ be a trace. A sequence $S = \prec a_1 \ldots a_k \succ$ is a "summary" of $T$ if and only if (a) $k \leq m$ and (b) for each $c_i \in T$ there is an $a_j \in S$ such that either $c_i = a_j$ or $c_i \mapsto a_j$, i.e. $c_i$ belongs to a subtrace that was summarized to the virtual center $a_j$.*

There are several possible summarizations of a trace, each one corresponding to a different partitioning of the trace into subtraces and consequently producing different virtual centers. We are interested in summarizations that achieve high space reduction while keeping information loss minimal. We generalize these objectives into functions measuring "space reduction" and "information loss", as explained below.

The replacement of a subtrace $X$ by its virtual center $\widehat{X}$ results in *storage space reduction*, since less nodes are stored, but also in *loss of information*, since the original clusters are replaced by a "virtual center". We model the information loss of each original cluster $c \in X$ as its distance from the virtual center $\widehat{X}$ to which it has been assigned after summarization:

$$ILoss\_cluster(c, \widehat{X}) = dist(\widehat{c}, \widehat{X}) \tag{1}$$

where $dist(\widehat{c}, \widehat{X})$ is the distance between the label of the original cluster $\widehat{c}$ and that of the virtual center $\widehat{X}$.

The information loss for a cluster/node is now aggregated at the level of the trace, to which the node belonged. The space reduction is also defined for traces.

**Definition 6 (Information Loss).** *Let $T$ be a trace and $S$ be a summary of this trace. The "information loss" of $T$ towards $S$ is:*

$$ILoss\_trace(T, S) = \sum_{c \in T} ILoss\_cluster(c, a_c) \tag{2}$$

*where $a_c \in S$ corresponds to either the virtual center to which c is mapped after the summarization or to the cluster c itself. In the latter case, $ILoss\_cluster(c, a_c) = 0$.*

**Definition 7 (Space reduction).** *Let $T$ be a trace and $S$ be a summary of this trace. The "space reduction" of $T$ towards $S$ is the decrease in the number of nodes and edges that need to be stored:*

$$\begin{aligned} SReduction\_trace(T, S) &= \tfrac{(|T|-|S|)+(|T|-1-(|S|-1))}{|T|+|T|-1} \\ &= \tfrac{2\times(|T|-|S|)}{2\times|T|-1)} \approx \tfrac{|T|-|S|}{|T|} \end{aligned} \tag{3}$$

*where $|T|$ is the number of nodes in $T$ and $|T|-1$ the number of edges among its nodes (similarly for $S$).*

This definition is similar to the definition of"compaction gain in [6].

Next, we define the "fingerprint" of a trace as a summary, the virtual centers of which are proximal to the original cluster labels, subject to a distance upper boundary $\tau$, so that the information loss effected through the replacement of a cluster by a virtual center is kept low.

**Definition 8 (Fingerprint for a Trace).** *Let $T$ be a trace and $S$ be a summary of $T$. $S$ is a "fingerprint" of $T$ if and only if:* C1 *For each node $c \in X$ that is replaced by a virtual center $a \in S$ it holds that $dist(\widehat{c}, a) \leq \tau$ and* C2 *for each (sub)trace $\prec c_1 \cdot \ldots \cdot c_k \succ$ of $T$ that has been summarized into a single virtual center $a$ it holds that $\forall i = 1 \ldots, k-1 : dist(\widehat{c_i}, \widehat{c_{i+1}}) \leq \tau$.*

By this definition, $S$ is a fingerprint of $T$ if it has partitioned $T$ into subtraces of clusters that are similar to each other (condition *C2*) and each such subtrace has a virtual center that is close to all its original nodes (condition *C1*).

Once traces are summarized to fingerprints, the Evolution Graph can also be summarized, resulting in space reduction and information loss at the graph level.

**Definition 9.** *Let $EG$ be an Evolution Graph and $\mathcal{T}_{EG}$ be its traceset. For each trace $T \in \mathcal{T}_{EG}$, let $S_T$ be its fingerprint (Definition 8), subject to a threshold $\tau$ on the distance among centroids. The set $\mathcal{S}_{EG} := \{S_T | T \in \mathcal{T}_{EG}\}$ is the "fingerprint of the Evolution Graph". It effects a space reduction $SR(EG, \mathcal{S}_{EG})$ and an information loss $IL(EG, \mathcal{S}_{EG})$:*

$$SR(EG, \mathcal{S}_{EG}) = \sum_T SReduction\_trace(T, S_T) \tag{4}$$

$$IL(EG, \mathcal{S}_{EG}) = \sum_T ILoss\_trace(T, S_T) \tag{5}$$

We next present the algorithm `batchFINGERPRINT` that creates the fingerprint of an Evolution Graph by partitioning traces in such a way that their fingerprints can be built. This algorithm requires that the Evolution Graph is first constructed and stored as a whole. Then, we present an online algorithm, `incFINGERPRINT`, that builds the fingerprints of the traces incrementally as new cluster transitions are detected. In this case, the fingerprint is built directly, without requiring the construction of the Evolution Graph first.

## 4.2 Batch Summarization of the Graph

`batchFINGERPRINT` summarizes an Evolution Graph $EG$ by identifying its traces, building a fingerprint for each trace and substituting the traces in $EG$ with their fingerprints. `batchFINGERPRINT` satisfies the two conditions of Definition 8 by applying two heuristics on each (sub)trace $T$:

- *Heuristic A:* If $T$ contains adjacent nodes that are in larger distance from each other than $\tau$, then the pair of adjacent nodes $c, c'$ with the maximum distance is detected and $T$ is then partitioned into $T_1, T_2$ so that $c$ is the last node of $T_1$ and $c'$ is the first node of $T_2$.
- *Heuristic B:* If $T$ satisfies condition *C2* but contains nodes that are in larger distance from the virtual center than $\tau$, then $T$ is split as follows: The node $c$ that has the maximum distance from $vcenter(T)$ is detected and $T$ is partitioned into $T_1, T_2$ so that $c$ is the last node of $T_1$ and its successor $c'$ is the first node of $T_2$.

*Heuristic A* deals with violations of condition *C2* and *Heuristic B* deals with violations of condition *C1* for (sub)traces that already satisfy *C2*. We show the algorithm in Fig. 3.

`batchFINGERPRINT` creates a fingerprint of the Evolution Graph by traversing the graph, extracting its traces (line 1, condition *C2*) and summarizing each of them (line 4). The "produced" fingerprints of the traces are added to the fingerprint graph $FEG$ (line 5). This operation encapsulates the attachment of a summarized trace to the graph by redirecting the ingoing/ outgoing edges of the original trace towards the ends of the summarized trace.

`batchFINGERPRINT` invokes $summarize\_HeuristicA$ which recursively splits the trace into subtraces according to *Heuristic A* until *C2* is satisfied. If the trace consists of only one node, then this node is returned (line 1). Otherwise, we test whether the trace contains nodes whose labels are further off each other than the threshold $\tau$ (line 2). If

```
batchFINGERPRINT($EG$)
Input: the Evolution Graph $EG$
Output: $FEG$, a fingerprint of $EG$
1. traverse the $EG$ and extract its traces into $\mathcal{T}$;
2. $FEG = \emptyset$;
3. for each trace $T \in \mathcal{T}$ do
4.     $FT = summarize\_HeuristicA(T)$;
5.     $FEG.addTrace(FT)$;
6. end-for

$summarize\_HeuristicA(T)$
Input: a trace $T$
Output: a fingerprint of the trace
1.     if $|T| == 1$ then return $T$;
2.     if $C\_b$ is not satisfied then
3.         find $c \in T$ such that
             $\forall (y,z) \in T\_1 : dist(y,z) < dist(c, c\_next)$ and
             $\forall (y,z) \in T\_2 : dist(y,z) < dist(c, c\_next)$ ;
4.         split T into $T\_1 = \prec c\_1, \ldots, c \succ$ and $T\_2 = \prec c\_next, \ldots, c\_k \succ$;
5.         $FT\_1 = summarize\_HeuristicA(T\_1)$;
6.         $FT\_2 = summarize\_HeuristicA(T\_2)$;
7.         return $\prec FT\_1 \cdot FT\_2 \succ$;
8.     else return $summarize\_HeuristicB(T)$;
9.    endif

$summarize\_HeuristicB(T)$
Input: a trace $T$
Output: a fingerprint of the trace
1.  $v = vcenter(T)$;
2.  if $\forall y \in T : dist(y,v) < \tau$ then return $v$; // Condition $C1$
3.  else
4.     find $c \in T$ such that $dist(c,v) = \max\{dist(y,v) | y \in T\}$;
5.     split $T$ into $T\_1 = \prec c\_1, \ldots, c \succ$ and $T\_2 = \prec c\_next, \ldots, c\_k \succ$;
6.     $FT\_1 = summarize\_HeuristicB(T\_1)$;
7.     $FT\_2 = summarize\_HeuristicB(T\_2)$;
8.     return $\prec FT\_1 \cdot FT\_2 \succ$;
```

**Fig. 3.** batchFINGERPRINT for offline summarization of the Evolution Graph

*C2* is satisfied, then $summarize\_HeuristicB$ is invoked (line 8): It checks for condition *C1* and returns the fingerprint of the (sub)trace input to it. If *C2* is violated, the trace is partitioned according to ***Heuristic A*** (lines 3,4) and $summarize\_HeuristicA$ is invoked for each partition (lines 5, 6). Finally, the summarized (sub)traces are concatenated (line 7) and returned. This concatenation operation restores or redirects the edges across which the split (line 4) was performed.

The recursive function $summarize\_HeuristicB$ operates similarly. It takes as input a (sub)trace $T$ that has more than one nodes and satisfies condition *C2*. It builds the virtual center for $T$ according to Definition 4. It then checks condition *C1* by comparing the distance of the virtual center from each node to $\tau$ (line 2). If $\tau$ is not exceeded, the virtual center is returned (line 3). Otherwise, $T$ is split at the node that has the highest distance from the virtual center, according to ***Heuristic B*** (lines 5, 6). The $summarize\_HeuristicB$ is invoked for each partition (lines 7, 8). The returned fingerprints are concatenated into the fingerprint of $T$.

### 4.3 Incremental Summarization

The batch summarization algorithm of Fig. 3 requires as input the complete Evolution Graph, before building its fingerprint. This is resource-intensive, since the graph is growing continuously. We have therefore designed `incFINGERPRINT`, an algorithm that summarizes the traces incrementally and does not require the a priori construction of the Evolution Graph. We show `incFINGERPRINT` in Figure 4.

```
incFINGERPRINT($FEG$)
Input: $FEG$ // the fingerprint built so far
    $\zeta$ // the most recent clustering, build at timepoint $t\_i - 1$
    $\xi$ // the current clustering, build at the current timepoint $t\_i$
Output: $FEG$ // the updated fingerprint
1.  $E\_i$=MONICtransitions($\zeta, \xi$);
2.  for each edge $e = (x, y) \in E\_i$ do
3.      if $e.extTrans \neq$ "survival" then
4.          $FEG.addNode(y)$;
5.          $FEG.addEdge(e)$;
6.      else if $dist(x.label, \widehat{y}) \geq \tau$ then // $C2$ is violated
7.          $FEG.addNode(y)$;
8.          $FEG.addEdge(e)$;
9.      else
10.         $v = vcenter(x, y)$;
11.         $FEG.replaceNode(x, v)$;
12.     endif
13. end-for
14. return $FEG$;
```

**Fig. 4.** `incFINGERPRINT` for online construction and summarization of the Evolution Graph

`incFINGERPRINT` invokes MONIC (line 1), which compares the current clustering $\xi$ (timepoint $t_i$) to the most recent one $\zeta$ (timepoint $t_{i-1}$), identifies the cluster transitions and returns them as a set $E_i$ of labeled edges, according to Subsection 3.2. The source of each edge corresponds to a node that is already in the fingerprint graph $FEG$. It is stressed that MONIC operates on the clusterings rather than the cluster labels retained in the nodes of the fingerprint graph. So, from line 2 on, `incFINGERPRINT` transfers information about the detected transitions in the $FEG$, summarizing survivals wherever possible. The result is an already summarized version of the Evolution Graph.

For each edge $e = (x, y)$, `incFINGERPRINT` examines whether $e$ is a survival transition (line 3), i.e. whether $e$ is part of a trace. If not, $FEG$ is expanded by adding the cluster $y$ and the edge $e$ (lines 4, 5). We do not add the whole cluster; we only retain its label (cf. Subsection 3.1).

If $e = (x, y)$ does belong to a trace, `incFINGERPRINT` checks whether the labels of $x$ and $y$ are similar to each other, according to condition *C2* of Definition 8 (line 6). Since cluster $x$ has already been added to $FEG$, we access its label $x.label$ directly, while the label of cluster $y$ must be computed as $\widehat{y}$. If condition *C2* is not satisfied, the $FEG$ is expanded by $y$ and $e$ as before. If finally, *C2* is satisfied, then $y$ and $e$ do not need to be added to $FEG$. Instead, $x$ and $y$ are summarized into their virtual center $v$ (line 10) and the node $x$ is replaced by $v$ (line 11). This means that all edges pointing to $x$ are redirected to $v$.

`incFINGERPRINT` not need to check for condition *C1*, since the distance of the virtual center of *two* nodes is less than the distance between the two nodes as a whole; the latter is less than $\tau$ by virtue of condition *C2*. This algorithm operates locally, treating pairs of adjacent nodes only, instead of whole traces. However, it has the advantage of not requiring the a priori construction of the Evolution Graph.

## 5 Experimental Results

The goal of our experiments is to measure the space reduction and information loss for different values of the centroid similarity threshold $\tau$ (c.f. Definition 8) that governs the summarization process.

### 5.1 Datasets

We experimented with two numerical datasets, the *Network Intrusion dataset* and the *Charitable Donation dataset*, used also in the stream experiments of [2], and with the document set ACM H2.8 used in the experiments of MONIC [13]. The first dataset is *rapidly evolving*, the second one is *relatively stable*, while the third one evolves in an *unbalanced* way - one of the classes grows faster than the others.

The *Network Intrusion dataset* (KDD Cup'99) contains TCP connection logs from two weeks of LAN network traffic (424,021 records). Each record corresponds to a normal connection or an attack. The attacks fall into four main categories: DOS, R2L, U2R, and PROBING. So, we set the number of clusters to 5, including the class of normal connections. We used all 34 continuous attributes for clustering and removed one outlier point, as in [2]. We turned the dataset into a stream by sorting on the data input order. We assumed a uniform flow in speed of 2,000 instances per time period. For data ageing, we assumed a sliding window of 2 time periods/timepoints.

The *Charitable Donation dataset* (KDD Cup'98) contains information (95,412 records) on people who have made charitable donations in response to direct mailings. Clustering identifies groups of donors with similar donation behavior. Similar to [8], we used 56 out of the 481 fields and set the number of clusters to 10. As with the previous dataset, we used the data input order for streaming and assumed a uniform flow with 200 instances per time period. For data ageing, we used a sliding window of size 2.

The *ACM H2.8 subarchive* is the set of documents inserted between 1997 and 2004 in the ACM Digital Library, category H2.8 on "Database Applications". This dataset contains publications on (1) data mining, (2) spatial databases, (3) image databases, (4) statistical databases and (5) scientific databases. It further contains (6) uncategorized documents, i.e. those assigned in the parent class "database applications [only]". The subarchive consists of the documents whose primary (or secondary) class is one of these 6 classes (4,920 records). It evolves in an unbalanced way [13]: The category (1) is larger than all the others together and grows faster than the others. For the experiments in [13], only the title and a list of keywords were considered for each document. We have used the same vectors and the same clustering algorithm, bisecting $K$-means, for $K = 10$. We turned the data into a stream by using the publication date for the ordering of the records. We considered $n = 7$ timepoints corresponding to the 7 publication years from 1998[1] to 2004; the corresponding data batches have different sizes varying from 837 in 1998 to 617 in 2004. The size of the sliding window was set to 2. A cluster label consists of the terms appearing in more than 60% of the cluster's vectors.

## 5.2 Example traces and fingerprints

To highlight the behavior of the summarization algorithms, we depict here some traces from the *ACM H2.8 dataset* and their fingerprints, as produced by our summarization algorithms.

In 1998, we observe a new cluster with the label "information systems". Its trace is $trace(c_{1998_2}) = \prec c_{1998_2} c_{1999_6} c_{2000_3} \succ$, where the notation $c_{y_i}$ refers to the $i^{th}$ cluster of year $y$, with $i = 1 \ldots 9$ (cluster 0 is the garbage cluster) [2]. The cluster centroids contain the terms "information" and "system" with the following frequencies:
$\widehat{c_{1998_2}} =< information(0.96), system(0.61) >$,
$\widehat{c_{1999_6}} =< information(0.88), system(0.74) >$ and
$\widehat{c_{2000_3}} =< information(0.76), system(0.78) >$.
Both summarization algorithms condense this trace into a single virtual center. The batch algorithm creates this new node $v$ in one step:
$\widehat{v} =< information(0.87), system(0.71) >$,
while the incremental first summarizes $c_{1998_2}$ and $c_{1999_6}$ into a virtual center:
$\widehat{v_0} =< information(0.92), system(0.68) >$,
and then summarizes $v_0$ and $c_{2000_3}$ into a new virtual center:
$\widehat{v'} =< information(0.84), system(0.73) >$.

A further cluster that emerged in 1998 had the one-term label $< analysis(1.0) >$. In 1999, it was split into two clusters, one labeled $< mining(1.0), datum(0.74) >$ and one cluster with no label (garbage cluster). The former survived for two periods, thus resulting in the trace $\prec c_{1999_8} c_{2000_4} c_{2001_6} \succ$. The information delivered without summarization is the sequence $\prec c_{1998_9} c_{1999_8} c_{2000_4} c_{2001_6} \succ$; $c_{1998_9} \overset{\subsetneq}{\rightarrow} \{c_{1999_8}, \prec c_{2000_4} c_{2001_6} \succ\}$; the summarization delivers the fingerprint $c_{1998_9} \overset{\subsetneq}{\rightarrow} \{c_{1999_8}, \widehat{v}\}$ instead, where $\widehat{v}$ is the summary of the trace $\prec c_{2000_4} c_{2001_6} \succ$.

[1] The timepoint 1998 includes publications of both 1997 and 1998, since the former contains only a small number of publications.

[2] Cluster identifiers are generated by the clustering algorithm at each timepoint.

## 5.3 Space Reduction and Information Loss

In Fig. 5 we show the space reduction achieved by the batch and the incremental summarization methods for each dataset and for different values of the centroid similarity threshold $\tau$. As we can see from this figure, the two algorithms achieve similar space

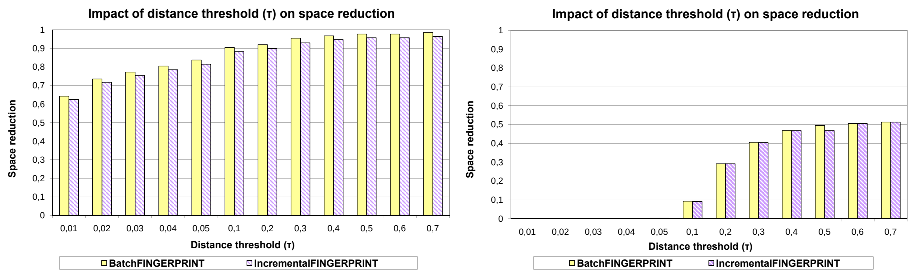

**Fig. 5.** Impact of threshold $\tau$ on space reduction for the Network Intrusion dataset (left) and the Charitable Donation dataset (right)

savings, although `incFINGERPRINT` shows slightly lower values for most values of $\tau$ in the *Network Intrusion* dataset. As expected, the space reduction increases for larger values of $\tau$, because less proximal centroids can be merged. The total space reduction for each dataset depend of course on the number of survivals per se: Among the total of 1,195 clusters/nodes generated for the Network Intrusion dataset, 400 nodes participate in traces; the space reduction values achieved by both algorithms are in the range [21%, 33%] of the total size of the Evolution Graph. The Evolution Graph of the Charitable Donation dataset contained 4,770 clusters, of which 614 were involved in traces; the space reduction over the whole graph were thus no more than 7%. For the ACM H2.8 subarchive, 24 out of 70 nodes were involved in traces, so that the space reduction over the whole graph ranged between 9% and 33%.

In Fig. 6 we depict the information loss effected upon the datasets when summarizing incrementally versus in batch. For the Charitable Donation dataset, the information loss incurred by the incremental algorithm is slightly higher than for the batch algorithm but follows the same curve for different values of $\tau$. For the Network Intrusion dataset, the performance difference is dramatic: While the batch algorithm achieves a very low information loss, the incremental algorithm performs very poorly. A possible explanation for the poor performance of `incFINGERPRINT` in the Network Intrusion dataset is the volatility of the dataset: It is likely that the survived clusters were unstable and not very similar to each other. Hence, `incFINGERPRINT` produced virtual centers that were not very close to the original pairs of centroids, while `batchFINGERPRINT` managed to build better virtual centers among multiple adjacent centroids.

In Fig. 7 we show the joint curves of space reduction and information loss for the two datasets and for different values of the centroid similarity threshold $\tau$.

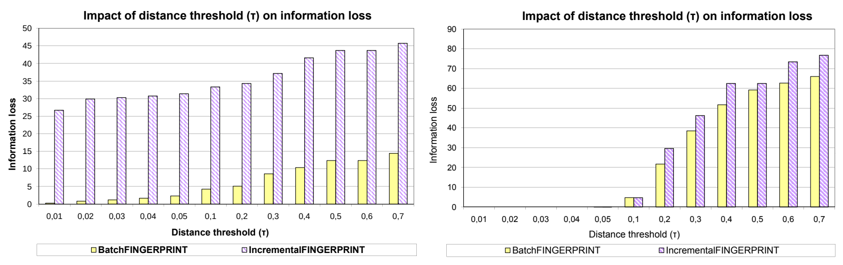

**Fig. 6.** Impact of threshold $\tau$ on information loss for the Network Intrusion dataset(left) and the Charitable Donation dataset (right)

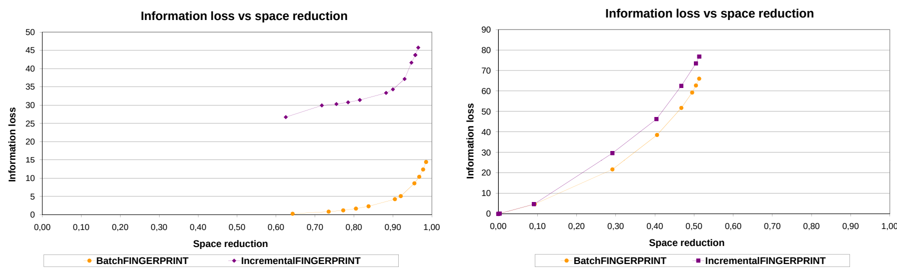

**Fig. 7.** Correlation between information loss and space reduction for Network Intrusion dataset (left) and the Charitable Donation dataset (right) for different values of $\tau$

## 6 Conclusions and Outlook

We have studied the effective summarization of cluster changes over an evolving stream of data. We modeled cluster transitions in a graph structure, the Evolution Graph, and proposed two algorithms that summarize it into a "fingerprint". A fingerprint is a condensed representation, in which less informative cluster transitions are suppressed. We derived functions that measure the effected information loss and space reduction and we presented heuristics that drive the summarization process. One of our algorithms summarizes the Evolution Graph as a whole, while the other creates the graph's fingerprint incrementally, during the process of cluster transition discovery. We have run experiments on three real datasets and have seen that incFINGERPRINT achieves similar space reduction to batchFINGERPRINT, but the information loss may be much higher depending on the volatility of the dataset.

The batch algorithm batchFINGERPRINT shows better performance comparing to the incFINGERPRINT algorithm, but it requires the whole dataset of transitions as an input. A hybrid summarization algorithm using both an online and an offline component is worth pursuing. Another interesting direction is modeling and investigation of the impact of the quality and stability of the original clustering on the summarization process. In this work, we have concentrated on the summarization of cluster survivals. A survival is the transition of a cluster to a similar successor cluster. Instead of placing constraints on the similarity among clusters, we want to study models of information loss for the summarization of arbitrary cluster transitions, so that only the most informative changes are delivered to the end-user.

## 7 Acknowledgments

Irene Ntoutsi is supported by an Alexander von Humboldt Foundation fellowship for postdocs (http://www.humboldt-foundation.de/). Part of this work has been conducted while I. Ntoutsi was at the Department of Informatics, University of Piraeus, Greece supported by the Heracletos program.

## References

1. Aggarwal, C.C.: On change diagnosis in evolving data streams. IEEE Trans. Knowl. Data Eng. 17(5), 587–600 (2005)
2. Aggarwal, C.C., Han, J., Wang, J., Yu, P.: A framework for clustering evolving data streams. In: VLDB (2003)
3. Aggarwal, C.C., Yu, P.S.: A framework for clustering massive text and categorical data streams. In: SIAM Data Mining Conf. (2006)
4. Bartolini, I., Ciaccia, P., Ntoutsi, I., Patella, M., Theodoridis, Y.: A unified and flexible framework for comparing simple and complex patterns. In: ECML/PKDD 2004 (2004)
5. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: SDM06 (2006)
6. Chandola, V., Kumar, V.: Summarization – compressing data into an informative representation. In: ICDM (2005)
7. Chen, Y., Tu, L.: Density-based clustering for real-time stream data. In: KDD. pp. 133–142 (2007)
8. Farnstrom, F., Lewis, J., Elkan, C.: Scalability for clustering algorithms revisited. SIGKDD Explorations 2(1), 51–57 (2000)
9. Gama, J.: Knowledge Discovery from Data Streams. CRC Press (2010)
10. Ipeirotis, P.G., Ntoulas, A., Cho, J., Gravano, L.: Modeling and managing content changes in text databases. In: ICDE. pp. 606–617 (2005)
11. Kalnis, P., Mamoulis, N., Bakiras, S.: On discovering moving clusters in spatio-temporal data. In: SSTD. pp. 364–381 (2005)
12. Mei, Q., Zhai, C.: Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In: KDD. pp. 198–207 (2005)
13. Spiliopoulou, M., Ntoutsi, I., Theodoridis, Y., Schult, R.: Monic: Mdeling and Monitoring Cluster Transitions. In: KDD. pp. 706–711 (2006)
14. Yang, H., Parthasarathy, S., Mehta, S.: A generalized framework for mining spatio-temporal patterns in scientific data. In: KDD05 (2005)
15. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An efficient data clustering method for very large databases. In: SIGMOD96. pp. 103–114 (1996)