INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

University of Alberta

Data Stream Summarization Methodology

by

Samer Nassar 

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the

requirements for the degree of *Master of Science*

Department of Computing Science

Edmonton, Alberta
Spring 2005

"... certainly God is conscienced by, from the people, the ones with knowledge... "

To God: the most gracious, the most merciful

To my best friend and first teacher: my father
To my best compassionate and first care: my mother
To my bests patience and joy: my sister and my brother

ABSTRACT

Knowledge discovery from multidimensional data streams requires a method for effectively maintaining a historical "collection" of summaries of past windows such that historical stream queries can be answered. We present the Stream Summary Store method. In this method, 'similar' spatial summary objects that summarize points in different windows are approximated by one approximate spatio-temporal summary object. A stream summary store is a collection of approximate spatio-temporal summary objects that consume a certain space budget. We outline various policies for satisfying the space budget constraint during the summarization of the data stream, and focus on policies that utilize prominent data mining and compression techniques (Clustering and Signal Compression) to further reduce the space consumption of the summary store. An extensive experimental evaluation shows that our methodology for constructing approximate spatio-temporal summary stores coupled with well-known Clustering and Signal Compression techniques significantly outperforms existing methods for summarizing multidimensional data streams.

ACKNOWLEDGEMENT

Professor Jörg Sander certainly taught me great concepts: from challenging problem formulation to creative comparative analysis to elegant presentation of scientific ideas. To you Professor Sander, I say thank you for supervising my work, thank you for inspiring me about Clustering and Data Streams, and thank you for challenging me with publishing in top conferences. I was certainly privileged. It was a great intellectual journey. Thank you.

Professor Russell Greiner certainly ignited my interest in Artificial Intelligence. To you Professor Greiner, I say thank you for your joy and unwavering help. It was fantastic to know that I could always stop by your office for extra advice. Thank you.

Professor Piotr Rudnicki certainly anchored my interest in Computing Science. To you Professor Rudnicki, I say thank you for teaching me from your admirable knowledge. It was very enjoyable to ponder concepts computationally. Thank you.

To my colleagues in the Data Base group Stanley Oliveira, Reza Sherkat, Jianjun Zhou, and others, thank you for thought provoking discussions and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

- B: a data bubble object
- $B_{over\text{-}filled}$: an over filled data bubble
- $B_{under\text{-}filled}$: an under filled data bubble
- CF: clustering features
- D: dimensionality of a data space
- E : an episode in the life history of an (ε,δ)-approximate spatio-temporal summary
- LS: linear sum of points summarized by a spatial summary object
- n: number of points summarized by a spatial summary object
- p: a multi dimensional object
- R: multi dimensional data stream
- rep: representative
- S: spatial summary object
- SE: summarization error of an (ε,δ)-approximate spatio-temporal summary
- SS: square sum of points summarized by a spatial summary object
- SS^W : set of spatial summary objects that summarize points in window W
- $stdev$: standard deviation of points summarized by a spatial summary object
- t: time stamp
- var: variance of a points summarized by a spatial summary object
- WS_R: window based summarization of a multi dimensional stream R
- β: summarization index of a spatial summary object
- ε: error level in the variance of a spatial summary object

- δ: error level in the mean of a spatial summary object
- $\hat{S}$: a set of spatial summary objects $\{S_1,...,S_k\}$
- S^W: a spatial summary object that summarizes (some of the) points in a window W
- Ψ_S: spatial configuration of a spatial summary object S
- $\Psi_{\varepsilon,\delta}^{\hat{S}}$: (ε,δ)-approximate spatial configuration of a set $\hat{S}$
- $\xi_{\varepsilon,\delta}^{\hat{S}}$: an (ε,δ)-approximate spatio-temporal summary of a set $\hat{S}$

CHAPTER ONE

INTRODUCTION

Knowledge discovery from data streams has attracted the interest of many scientists, as data streams are quickly becoming a prominent computational model and emerging as an application in many extraterrestrial and planetary fields including astrophysics, financial markets, and phone and web services.

Currently, knowledge discovery from data streams requires a method for effectively summarizing incoming windows and efficiently maintaining a historical "collection" of summaries of past windows such that queries pertaining to these collections of summaries of past windows can be effectively answered. Achieving effective answers is particularly important in conducting long term analysis of clusters evolution, where the emergence of several clusters and subsequent changes to them during the stream can be analyzed over a long period of time to make better decisions during the cluster mining process instead of relying on a limited short time period.

In this thesis, to achieve an effective window based summarization of a multi-dimensional data stream, we present the stream summary store methodology. The thesis is organized as follows. In chapter 2, we give a brief background of the data summarization method that we use for compressing hierarchical clusters: data bubbles. To develop our stream window based summarization methodology, we begin by designing in chapter 3 an effective and efficient dynamic summarization under a database environment. In chapter 4, we formulate the problem of effective window

based summarization of multi-dimensional data stream and present our stream summary store methodology that solves this problem. We design an approximate spatio-temporal summary object that compresses 'similar' spatial summary objects that summarize points in different windows. A stream summary store that consumes a certain space budget is constructed using approximate spatio-temporal summary objects. Finally, we design various policies for satisfying the space budget constraint during the summarization of the data stream, and focus on policies that utilize prominent data mining and compression techniques (Clustering and Signal Compression) to reduce the space consumption of the summary store. By effectively integrating summarizations of many stream windows, our stream summary store is the first step towards achieving analysis of cluster evolution in multi-dimensional data streams. In chapter 6, we present the conclusions of our work: maintaining incremental data bubbles and stream summary stores. The bibliography is presented in chapter 7.

The contributions of the thesis are:

1. A method to speed-up the construction of data summaries during the assignment of points to the representatives of the data summaries.
2. An efficient and effective method for incrementally maintaining a given number of data bubbles in a dynamic database environment with insertions and deletions.
3. A general framework for summarizing multi-dimensional data streams.

4. A novel approximate spatio temporal summary that is capable of effectively approximating several 'similar' spatial summary objects that summarize points in different stream windows.
5. Dynamic stream summary stores that satisfy a given storage space budget constraint and are capable of providing effective answers to queries of summaries of historical stream windows.

CHAPTER TWO

DATA BUBBLES

In recent years and with the massive increase in the size of databases and the emergence of very large data streams, the development of scalable clustering algorithms has received a lot of attention in KDD. One approach for scaling up a clustering algorithm is to reduce its runtime such that it can be applied very quickly to large data sets and still effectively uncover the clustering structure within acceptable runtime limits. This reduction in runtime can be achieved by applying the clustering algorithm to only a *summary* of the database instead of the whole database. In data summarization methods such as Data Bubbles [5] and BIRCH [27], the database is partitioned into a small number of subsets, where each subset represents its elements by a number of sufficient statistics. A modified version of the preferred clustering algorithm can be applied then to those data summarizations to detect the interesting patterns.

Previously it has been shown that for hierarchical clustering algorithms, the so-called *data bubbles* [5] are much more effective than basic clustering features $CF=(n,$

LS, SS), where *LS* is the linear sum of *n* points and *SS* is their square sum, as proposed, e.g., for BIRCH[27]. Data bubbles summarize a set of *n* points by "compressing" the points into special sufficient statistics that are required for effective hierarchical clustering based on data summarizations. Data bubbles have been evaluated in [5], using OPTICS [2], and were shown to reduce the runtime of OPTICS dramatically while still producing high-quality hierarchical clustering structures.

A data bubble has been defined as follows [5]:

Definition 1. *A **data bubble** B for a set of points* $X = \{X_i\}$, $1 <= i <= n$ *is a tuple*

$$B = (rep, n, extent, nnDist)$$

where

- *rep is a representative, defined as the mean of the points in X*
- *n is the number of points in X*
- *extent is the radius of B around rep that encloses majority of the points in X*
- *nnDist(k,B) is a function that estimates the average k nearest neighbour distances in B* □

Although the information in a data bubble is more specialized than the basic sufficient statistics (*n, LS, SS*), it has been shown in [5] that the representative *rep*, the *extent*, and assuming a uniform distribution of points within a data bubble, the average nearest neighbor distances *nnDist(k,B)* can be easily derived from *n, LS, SS*. In this thesis, we use data bubbles because our intended application is analyzing hierarchical clusters, and data bubbles have been shown in [5] to be suitable for this

task. In the next chapter, we present a new method for incrementally maintaining data bubbles.

CHAPTER THREE

INCREMENTAL AND EFFECTIVE DATA SUMMARIZATION FOR DYNAMIC HIERARCHICAL CLUSTERING [18]

3.1 Motivation

Knowledge Discovery in Databases (KDD) has been instrumental in uncovering useful patterns hidden in very large databases, improving the understanding of these patterns, and aiding in making better decisions related to the databases. Detecting patterns effectively and efficiently in real world databases is a challenging task since these patterns usually reside in large amounts of high dimensional and noisy data. As time goes by, the data distribution and the underlying clustering structure may change whereby previously uncovered patterns may become obsolete. The ability of a data mining technique to detect and react quickly to dynamic changes in the data patterns is highly desirable.

Clustering is one of the most prominent and frequently used data mining techniques in KDD. The main goal of a clustering algorithm is to partition a set of data points into groups such that similar points belong to the same group and dissimilar points belong to different groups. There are two main kinds of clustering algorithms: partitioning and hierarchical. Partitioning algorithms like k-means [17] create k partitions of the points. Hierarchical clustering algorithms like the Single-

Link method [22] or OPTICS [2] compute a representation of the possible hierarchical clustering structure of the database in the form of a dendrogram or a reachability plot from which clusters at various resolutions can be extracted, as has been shown in [20].

Various dynamic updates of deletions and insertions to very large databases add new challenges to the clustering task by possibly changing the underlying data distribution and the associated clustering structure over time. The naïve approach is to reapply the data mining algorithms and extract the hidden patterns every time following a certain fraction of updates to the database. However, this approach is prohibitively slow for fast changing and large databases, especially if an up-to-date clustering structure is required frequently, e.g., in order to detect the changes in the data distribution after a small fraction of updates occur and important decisions are based on the current data distribution. For example, for effective marketing and early detection of changing purchasing patterns, or fraudulent transactions on debit cards, it is very important to maintain a large history of transactions for all current customers/subscribers, in order to detect possible changes in the clustering structures, which could indicate possible changes in the customer/subscriber behaviour.

There are two main strategies to address the problem of incremental clustering in a database environment. In the first strategy, a specialized incremental clustering algorithm is designed to directly handle dynamic changes in the database. In the second strategy, a data summarization technique is developed and used to compress the database incrementally, and then a slightly modified, standard clustering algorithm is subsequently applied to the generated data summarizations.

Unlike the first strategy that typically invents yet another "new" incremental algorithm (with possible unclear properties) for a particular application, the second strategy is more flexible and generic as it allows the application of a broad range of existing standard clustering algorithms (hierarchical and partitioning) to the data summaries. The adaptation of a standard clustering algorithm to data summarization typically requires only minor modifications, as has been shown in [5]. It also has the advantage that the data summaries can be used for other data mining tasks such as computing approximate statistics of data sets or quickly approximating the number of objects in a database within certain attribute ranges of interest.

In this chapter, we expand the second approach and propose a scheme to incrementally maintain data summaries of a dynamic database, i.e., we enhance data summarizations to become incremental and capable of adapting to insertions and deletions into a database. We choose the so-called data bubbles proposed in [5] for this task over the clustering features as proposed for BIRCH [27], which is another data summarization method that could be used for handling dynamic changes. We choose to enhance data bubbles because the intended applications of the achieved incremental data summarization include obtaining effective *hierarchical* clustering structures very quickly for large changing databases, and that it has been shown in [5] that data bubbles outperform clustering features significantly in this respect.

In this chapter, we show that our incremental data summarization method is effective in handling dynamic changes to a hierarchical clustering structure because the majority of the data bubbles can adapt to both insertions and deletions without rebuilding them. The data bubbles partition the space into sub-regions. Thus, during

the dynamic updates to a data base, the incremental data bubbles are able to detect the local effects of insertions and deletions in these sub-regions more easily than comparing all of the current distribution to the previous distribution of the database prior to the most recent insertions and deletions.

Furthermore, by using a measure of the compression quality, we can identify the data bubbles that still compress their points well following the insertions and the deletions. The sub-regions that cause some of the data bubbles to have low compression quality -possibly due to changes in the underlying data distribution- will result in data bubbles that require rebuilding. Typically, the number of these sub-regions is small and thus the majority of the data bubbles can adapt easily to even very large numbers of insertions and deletions.

The contributions of this chapter are:

1. A method to speed-up the incremental construction of data summaries by utilizing triangle inequalities when assigning points to the representatives of the data summaries.
2. A scheme for incrementally maintaining a given number of data bubbles in a dynamic database environment with insertions and deletions.
3. A quality measure for identifying the incremental data bubbles that degrade the clustering structure most significantly.
4. Efficient and synchronized merge and split operations for rebuilding incremental data bubbles that have low compression quality in order to improve the effectiveness of the over all data summarization and consequently

the quality of the analytical results obtained from the database using only the data bubbles such as hierarchical clustering structures.

3.2 Related Work

The problem of incremental clustering has been studied by many scientists. In this section, we discuss some of the proposed algorithms. There are several incremental clustering algorithms that do not use the data summarization technique but attempt to directly restructure the clusters to adapt to the dynamic changes of the dataset.

Chen et al. [7] propose the incremental hierarchical clustering algorithm GRIN for numerical datasets, which is based on gravity theory in physics. In the first phase, GRIN uses GRACE, which is a gravity-based agglomerative hierarchical clustering algorithm, to build a clustering dendrogram for the data set. Then GRIN restructures the clustering dendrogram before adding new data points by flattening and pruning its bottom levels to generate a tentative dendrogram. Each cluster in the tentative dendrogram is represented by the centroid, the radius, and the mass of the cluster (which is the number of points in the cluster). In the second phase, new data points are examined to determine whether they belong to leaf nodes of the tentative dendrogram. If a new point belongs to only one node, then it is inserted in that node. Else, the gravity theory is applied to determine the leaf node that the point belongs to, and the point is added to the selected leaf.

Ester et al. [10] present a new incremental clustering algorithm called IncrementalDBSCAN suitable for mining in a data-warehousing environment. IncrementalDBSCAN is based on the DBSCAN algorithm [9], which is a density based clustering algorithm. Due to its density-based qualities, in

IncrementalDBSCAN the effects of inserting and deleting objects are limited only to the neighborhood of these objects, where only these objects may undergo a change in their core property, while other objects retain their core property. IncrementalDBSCAN requires only a distance function and is applicable to any data set from a metric space. However, the proposed method does not address the problem of changing point densities over time, which would require adapting the input parameters for IncrementalDBSCAN over time.

Widyantoro et al. [25] present the agglomerative incremental hierarchical clustering (IHC) algorithm that also utilizes a restructuring process while preserving homogeneity of the clusters and monotonicity of the cluster hierarchy, where a homogenous cluster is a set of points with similar density (i.e. their distances to their closest neighbours are approximately equal), and the monotonicity of the clustering structure requires that the density of a cluster is always higher than the density of its parent in the clustering structure. New points are added in a bottom-up fashion to the clustering hierarchy, which is maintained using a restructuring process performed only on the regions affected by the addition of new points. The restructuring process repairs a cluster whose homogeneity has been degraded by eliminating lower and higher dense regions.

3.3 Speeding Up The Construction of Data Bubbles

In this section, we consider the problem of using incremental data bubbles to speed up *hierarchical* clustering of large databases. Previously it has been shown that for hierarchical clustering algorithms, the so-called *data bubbles* are much more effective than basic clustering features $CF=(n, LS, SS)$, where LS is the linear sum of n points

and *SS* is their square sum, as proposed, e.g., for BIRCH. Data bubbles have been evaluated in [5] using OPTICS [2], and were shown to reduce the runtime of OPTICS dramatically while still producing high-quality hierarchical clustering structures.

The method that has been proposed to construct data bubbles consists of the following two steps:

1. Retrieve randomly *s* points from the database as "seeds".
2. Scan the database, and assign each point in the database to the closest seed in the set obtained in step 1.

In step 2 of this construction algorithm, the closest seed of a data bubble to a point *p* has to be found. In a standard implementation, the distance between *p* and all the seeds has to be determined to make that decision. Although we assume that only a relatively small number of data bubbles is used to represent a database, these distance computations offer a big potential for optimization.

We propose to use triangle inequalities to reduce the runtime of constructing the data bubbles significantly. Relative to distance comparisons, distance calculations are computationally much more expensive.[1] The idea is to avoid these computationally expensive distance calculations by using the much cheaper distance comparisons when applying certain triangle inequalities. The method is based on the observation that the computation of certain distances between seeds and database points can be avoided if the pairwise distances between the seeds are known. A sufficient condition under which this observation applies is stated in the following lemma:

[1] Related techniques for pruning distance calculations using triangle inequalities have been successfully applied in the computation of similarity queries [4] and in *k*-means [8]

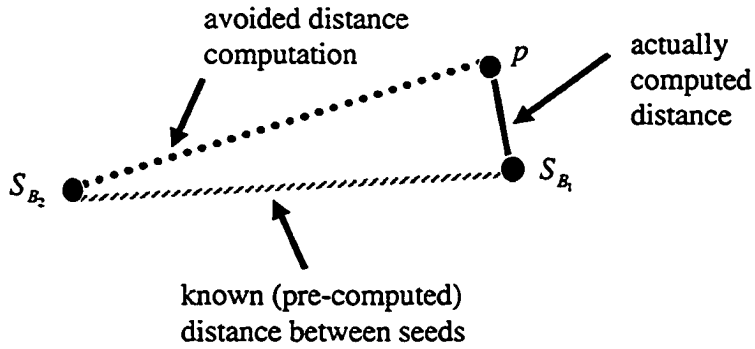

Figure 1. Pruning of Distance Computations

Lemma 1: *Let p be a database point, and let s_{B1} and s_{B2} denote the selected seeds of two data bubbles B_1 and B_2 respectively. If $dist(s_{B1},s_{B2}) \geq 2*dist(p,s_{B1})$, then $dist(p,s_{B1}) \leq dist(p,s_{B2})$.* □

The lemma is illustrated in figure 1. Assume we have pre-computed the distances among all the seeds selected in step 1 in the construction of data bubbles (once, prior to step 2). To determine which of the seeds is closer to a point p, we have to compute the distance between p and at least one of the seeds, say s_{B1}. Assuming that this distance is as depicted in figure 1, and the distance between s_{B1} and s_{B2} is larger than twice this distance, we can actually avoid the computation of the distance between p and s_{B2} since we can conclude using lemma 1 that s_{B2} cannot be closer to p than s_{B1}. To utilize the above lemma during the assignment of points to their closest seeds, we maintain a distance matrix that stores the distances among the seeds of all of the data bubbles. Typically, the overhead of computing distances among the data bubble seeds is low since the number of data bubbles is small and more than compensated by the huge fraction of distance calculations between database points and the seeds that are consequently avoided.

The assignment of a point p to the closest data bubble using the triangle inequalities proceeds as follows. First the distance of a database point p to the seed s_c of a randomly selected data bubble is computed. This seed is the current candidate data bubble for assigning the point to from the set of data bubbles. We try to prune the seeds s_i of all the other data bubbles without computing their distances to p by looking up the distances between s_c and s_i and applying Lemma 1. If all data bubbles can be pruned, then s_c is the closest seed to p. Otherwise, we attempt to find a closer seed to p by computing the distance to another un-pruned seed s_j. If s_j is closer to p than the previous s_c, then s_j becomes our new current candidate and we attempt to prune the remaining data bubbles in a similar fashion using the distance to the new candidate. This pruning and updating of the candidate seed is iterated until there is only one candidate seed left, which has to be the closest to the point p. The point p is assigned to the closest data bubble. The pseudo code for this procedure is depicted in Figure 2.

In the following presentation of the scheme for incrementally maintaining a set of data bubbles, we assume that we have initially constructed a set of data bubbles that summarize a large database of d-dimensional points following the description in the previous section. As indicated, the purpose of our data summarization is to be able to obtain a hierarchical clustering result very quickly for the whole database, based on the data bubbles. If the database is dynamic, new points are inserted and old points are deleted over time, possibly changing the underlying data distribution. We are interested in the updated clustering structure and hence the underlying data summarization after a set of updates during which $N\%$ points have been deleted and

```
set CandidateSeeds to the set of all seeds of data bubbles
select and remove a random seed s_c from CandidateSeeds
compute minDist = dist(p,s_c)
while CandidateSeeds is not empty
    for all s_i in CandidateSeeds
        look up the distance d_{s_i,s_c} between s_i and s_c
        if d_{s_i,s_c} >= 2 * minDist
            remove s_i from CandidateSeeds
    while CandidateSeeds is not empty
        select and remove a random seed
        compute dist(p,s_j)
        if dist(p,s_j) < minDist
            set s_c = s_j
            set minDist = dist(p,s_j)
            break
return s_c
```

Figure 2. Finding the Closest Data Bubble Seed for a Point p.

M% points have been inserted (where N and M are parameters that determine the amount of updates after which we want to inspect the changes in the hierarchical clustering structure).

The high-level description of our scheme for incrementally updating a set of data bubbles following a batch of updates to the underlying database is given in Figure 3. In a nutshell, the sufficient statistics of affected data bubbles are decremented when deleting the old points and incremented when inserting the new points. When deleting a point p, the sufficient statistic (n, LS, SS) of the data bubble B where p was previously assigned are updated to ($n-1$, $LS-p$, $SS-p^2$), whereas when inserting a point

1. Delete $N\%$ of the old points and decrease the sufficient statistics of the corresponding data bubbles.
2. Insert $M\%$ new points and assign them their closest data bubbles (using the improved assignment algorithm described in section 3.3).
3. Determine the compression quality of the data bubbles.
4. Rebuild data bubbles that have a low compression quality.

Figure 3. Overall Scheme of Maintaining Data Bubbles Incrementally.

p, the sufficient statistics (n, LS, SS) of the data bubble B that is closest to p are updated to ($n+1$, $LS+p$, $SS+p^2$).

After these updates, it is possible that some data bubbles do not represent their points well or lost all of their points such that the overall compression quality is poor, possibly resulting in a distorted clustering structure based on these data bubbles. In order to recover from structural distortions due to changes in the data distribution, we have to identify those data bubbles that significantly degrade the quality of the data summarization and re-build them quickly, while at the same time maintaining a given compression rate.

3.4 The Compression Quality of Data Bubbles

To achieve a high quality of an overall compression by the data bubbles, we need to distinguish "good" data bubbles that have a high quality of compression from data bubbles that have a low quality of compression. Since building data bubbles completely from scratch can be considered as a baseline algorithm that has been shown to perform well for hierarchical clustering [5], we can assume that when building data bubbles from scratch, the majority of the data bubbles has "good"

compression quality by construction (due to randomization effects we can not exclude to have a few data bubbles with a "bad" compression even in this case though). However, the important question is how to define and measure the compression quality of data bubbles.

Clustering features constructed by BIRCH [27] can be viewed as being incremental with respect to insertions only. These methods implicitly suggest, as a quality measure for clustering features, the diameter, or the standard deviation of the distances from the mean, by the way they construct and maintain the clustering features. Roughly speaking, clustering features can "absorb" points as long as the resulting diameter or a related measure does not exceed a given maximal value provided as an input parameter.

These statistics are all quantifying the "spatial extent" of the clustering feature, i.e., measuring a kind of radius around the mean into which the points compressed by the clustering feature fall. We argue that the spatial extent is not a suitable measure for the quality of data summarizations, especially in an incremental setting.

Solving the problem of "what are the clusters in a database?" often depends on the resolution at which we analyze the database. Hierarchical clustering algorithms try to leverage this problem by constructing a hierarchical representation of the data that can reveal clusters at different levels of resolution. Setting a threshold for the spatial extent of the data summarizations is equivalent to fixing a resolution at which the clusters can be found. This is already a severe limitation for a static database.

Moreover, setting a global threshold parameter for the spatial extent basically equalizes the extents of the clustering features and the data bubbles such that the data

space is split more or less equally among the data bubbles. However, it is not uncommon in many applications to have richer and denser substructures in some regions of the data space than in others, although the regions may occupy the same volume. Such important differences may not be detected if the number of data bubbles that are located in the area that contains the substructure is too low because the region (but not the number of points) covered by the substructure is relatively small compared to the specified extent parameter for the data bubbles. In dynamic databases where the data distribution may change over time, the clustering substructures can evolve at lower levels of a hierarchical clustering structure and go undetected if they are located within the allowed radius of a data bubble.

The measure that is much more significant for determining the quality of a data bubble is the number of points it summarizes relative to the total database size. Roughly (and vaguely) speaking, "good" data bubbles summarize not too many and not too few points.

On the one hand, potentially "bad" data bubbles summarize a large fraction of the total number of points. These data bubbles may easily span several substructures that are lost in a subsequent clustering of all the data bubbles, and thereby critically degrading the quality of the clustering result. In a dynamic setting, for instance, an over-filled data bubble can even arise when a new cluster appears in the database in an area that is not covered well by data bubbles (e.g., a previous noise region).

On the other hand, data bubbles that compress a very small fraction of the whole database are also not good in the sense that these data bubbles may become empty very quickly when all their points are deleted and no new points are inserted in the

regions that they span whereby they do not contribute much to the overall compression rate. These data bubbles do not directly influence the quality of the hierarchical clustering results based on data summarization. However, they may degrade the clustering result indirectly to some degree because they are in the sense "wasted" that it would be better to release their points (and assign them to the nearby data bubbles), and position their representatives elsewhere in the data space, where they can contribute more to the overall quality of the data summarization.

To capture the quality of a data bubble, we introduce the *data summarization index* β that we define to be the fraction of points in the database compressed by the data bubble.

Definition 2. *Given a data base D of N points and a set Ω of data bubbles that compress the points in D, the* ***data summarization index*** β_i *of a data bubble i that compresses n points is defined as* $\beta_i = \frac{n}{N}$ □

In order to determine which data bubbles have a low quality of compression, we know from our initial observation that when building data bubbles from scratch, the majority of the data bubbles have good compression. Thus, a data bubble has a bad compression if its fraction of points is significantly different from the majority of fractions of points in the data bubbles. The question is how to determine the β values that define "good" data bubbles. Even after a complete construction of the data bubbles from scratch, there is some significant variability in the number of points per data bubble due to different point densities in different regions of the data space. However, we can analyze the distribution of β values in order to determine which data bubbles have a good quality of compression and which do not.

In a set Ω of data bubbles that compress the database D, the β values of all the data bubbles follow a certain distribution. By analyzing the statistical properties of the mean and the standard deviation of this distribution, we recognize the outlier β values that identify the data bubbles that have significantly low compression quality and which require special handling in our scheme of incremental data summarization. Although we don't know the exact distribution of the β values, we can determine the outliers in the distribution by estimating the lower and upper boundaries of the β interval that characterizes the "good" data bubbles through using Chebyshev's Inequality theorem [24]. According to the theorem, if μ_X and σ_X are the mean and standard deviation of a random variable X, then for any positive constant k

$$P(|X-\mu_X|<k\sigma_X)\geq 1-\frac{1}{k^2}$$

Thus, the probability that a random variable will take on a value within k standard deviations of the mean of the distribution of its values is at least $1 - 1/k^2$ *regardless of the distribution.* By considering our data summarization index β as a random variable with the mean μ_β and the standard deviation σ_β and for a specific probability p, the value k can be determined as well as the upper (and lower) boundary of the region that contains at least $p\%$ of the β values. The upper boundary is $\mu_\beta + k\sigma_\beta$ (and the lower boundary is $\mu_\beta - k\sigma_\beta$).

A data bubble that compresses several substructures would contain a large fraction of points, its β value would be significantly larger than the average, and therefore its β value would be above the upper boundary $\mu_\beta + k\sigma_\beta$ (the β value would be located towards the right end of the distribution). On the other hand, β values that are significantly lower than the average β value are below the lower boundary $\mu_\beta - k\sigma_\beta$.

These low β values identify data bubbles that are (nearly) empty (i.e. they compress relatively very few or no points). Using these statistical boundaries in the distribution of the data summarization index, we distinguish three classes of data bubbles according to their compression quality.

Definition 3. *Given a data base D of N points and a set Ω of data bubbles that compress the points in D, let μ_β and σ_β be the mean and standard deviation of the distribution of the β values for all data bubbles in Ω. Given a probability p (where the corresponding k value is computed according to Chebyshev's Inequality), a data bubble B with the data summarization index β is called:*

- ***"good"** iff $\beta \in [\mu_\beta - k\sigma_\beta, \mu_\beta + k\sigma_\beta]$*
- ***"under-filled"** iff $\beta < \mu_\beta - k\sigma_\beta$*
- ***"over-filled"** iff $\beta > \mu_\beta + k\sigma_\beta$* □

Improving the quality of the over-filled data bubbles is immediately critical for providing a high quality data summarization of the given database. Although the under-filled data bubbles have a low compression quality, their effect on the hierarchical clustering structure is not as significant as the effect of the over-filled data bubbles. The under-filled data bubbles do not contribute significantly to the overall data summarization and in principle could remain as-is without attempting to improve their compression quality. Thus, we focus on improving the compression quality of the over-filled data bubbles through "splitting" them by migrating possible under-filled data bubbles, as explained next.

3.5 Maintaining Incremental Data Bubbles

The main objective of our incremental data summarization scheme is to efficiently improve the quality of the over-filled data bubbles since they degrade the compression quality most. A natural way to reduce the number of points in a data bubble is to reassign some of these points among more data bubbles, whereby an over-filled data bubble gives up some of its points to other data bubbles.

The naïve approach is to reassign some of the points in an over-filled data bubble to their next closest data bubbles (the closest data bubble of each of these points is the over-filled data bubble they are currently assigned to by construction). These next closest data bubbles are the surrounding neighbours of the over-filled data bubble. However, reassigning some of the points in the over-filled data bubble to (some) of its neighbouring data bubble is very likely to reduce the compression quality of these neighbouring data bubbles due to the following reasons.

When constructing a set of data bubbles to compress a given database, more seeds are likely to be selected from the dense regions in the data space due to the random seed selection process. Thus, typically data bubbles "share" dense regions, and the majority of the data bubbles have a good quality of compression. When the compression quality of a data bubble degrades from good (or even possibly under-filled) to over-filled, then the number of points it compresses has increased dramatically but not for other data bubbles. The over-filled data bubble has absorbed a large number of new points while its neighbours have not, which indicates that the neighbouring data bubbles are not close to the over-filled data bubble, i.e. the new points have appeared in a region that is not summarized by the neighbouring data bubbles.

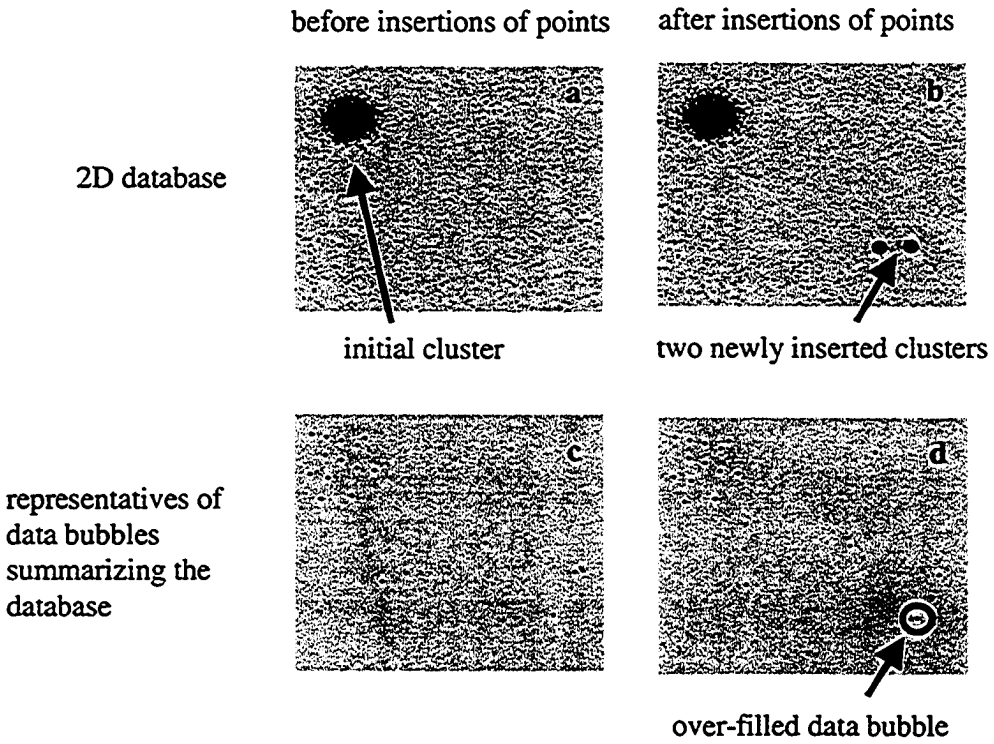

Figure 4. Over-filling of a Data Bubble by New Clusters.

In figure 4, we see an example of this over-filling effect. Given an initial database (part a), the seeds for the data bubbles (part c) are selected randomly during the construction phase with more seeds selected in the region of the cluster. When two new clusters are inserted far from the initial cluster (part b), there are few data bubbles in the vicinity of these new clusters, in this case only one such data bubble, and this data bubble becomes over-filled by absorbing these new clusters. Reassigning some of the points in the over-filled data bubble (identified by a circle in part d) to its neighbouring data bubbles would force the neighbouring data bubbles to absorb points that are located far away from the regions they compress, thereby significantly degrading their compression quality and distorting the net clustering structure.

We propose, instead, to position additional data bubbles. In our incremental data summarization scheme, we can not assume that we have access to an unlimited number of unused data bubbles that can be used for improving the compression quality of the over-filled data bubbles since "splitting" an over-filled data bubble requires positioning additional data bubbles in the vicinity of the center of the over-filled data bubble. We already know that the under-filled data bubbles have low β values and contain relatively few (or no) points that can be distributed among neighboring data bubbles without significantly affecting the quality of these neighboring data bubbles. Once the points of these under-filled data bubbles are redistributed, these data bubbles can be re-used. Thus, we can migrate them and reposition them in the vicinity of the centers of over-filled data bubbles to achieve the splitting of the over-filled data bubbles.

We can see an example of this migration of under-filled data bubbles in Figure 5. For a given database (part a), the set of data bubbles that summarizes the points in this database contains few data bubbles that are under-filled. Following the insertion of the two new clusters (part b), under-filled data bubbles (identified in circles in part c) are re-positioned and migrate to the region of the two new clusters (part d) to improve the compression of these new clusters.

In the current approach of incremental data bubbles for handling updates in the database, the quality of an over-filled data bubble $B_{over\text{-}filled}$ is improved by merging and ideally re-positioning an under-filled data bubble $B_{under\text{-}filled}$ to the vicinity of the center of $B_{over\text{-}filled}$ and "splitting" $B_{over\text{-}filled}$ into two new data bubbles B_1 and B_2. In the absence of an under-filled data bubble, we utilize enough data bubbles from the

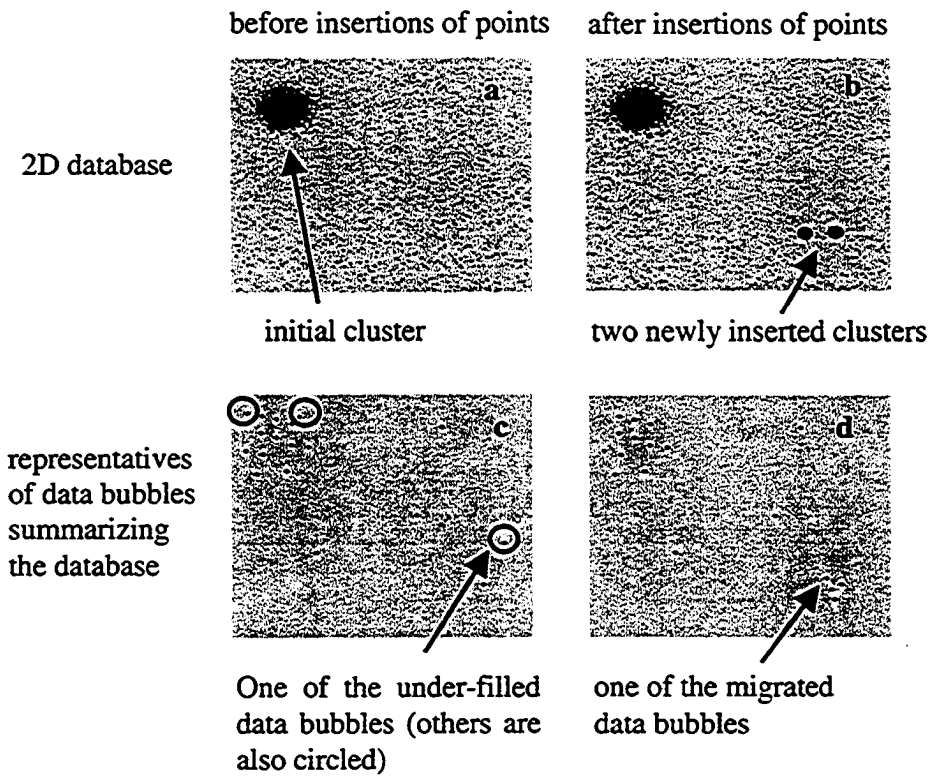

Figure 5. Migration of Under-filled Data Bubbles.

"good" data bubble subset to split all the over-filled data bubbles. We select the lowest quality data bubbles from the "good" subset to perform the splitting of all the over-filled data bubbles.

Figure 6 shows the pseudo code for this process. The quality of $B_{over\text{-}filled}$ is improved by first merging $B_{under\text{-}filled}$ and then splitting $B_{over\text{-}filled}$. During the merge phase, the points in the $B_{under\text{-}filled}$ are released and assigned to their next closest data bubbles thereby emptying $B_{under\text{-}filled}$. $B_{under\text{-}filled}$ is re-positioned to the region of $B_{over\text{-}filled}$ by selecting a new seed s_1 for it from the current points in $B_{over\text{-}filled}$. Next, $B_{over\text{-}filled}$ is assigned a new representative s_2 from its current points, and the points in $B_{\text{over-filled}}$ are distributed between the two newly selected representatives s_1 and s_2. We utilize the triangle inequalities mentioned above throughout the process of assigning a

1. Select a random under-filled data bubble $B_{under\text{-}filled}$ (if none exists, select the "good" data bubble with lowest quality in the "good" data bubbles set)
2. Free $B_{under\text{-}filled}$ by assigning its points to their next closest data bubble(s)
3. Migrate $B_{under\text{-}filled}$ to the region compressed by $B_{over\text{-}filled}$ by selecting a new seed s_1 for it from the points of $B_{over\text{-}filled}$
4. Select a new seed s_2 for $B_{over\text{-}filled}$ from the points of $B_{over\text{-}filled}$
5. Split $B_{over\text{-}filled}$ by reassigning its points between s_1 and s_2

Figure 6. Improving the Quality of an Over-filled Data Bubble.

point to its closest data bubble. The sequence of synchronized merging and splitting of data bubbles is repeated after updating the database with each batch of insertions and deletions.

3.6 Performance Evaluation

In this section, we perform an extensive evaluation of our scheme for achieving incremental data bubbles. The results show that our new method for incremental data summarization is suitable to be used with a clustering algorithm for mining hierarchical clustering structures very efficiently from dynamically changing databases, and that it is scalable and well suited for high dimensional data.

We first compare the adaptation of the data bubbles to insertions and deletions of points when using the fraction of points versus the extent as the measure of the compression quality. We perform a simple experiment where we demonstrate that if we use the extent of a data bubble as the quality measure instead of its relative number of points, the extent quality measure fails to produce a high quality of

compression while our quality measure does not. As figure 7 shows, we use a simple database that consists of two clusters before any insertions and deletions of points. During the insertions and deletions of points, the cluster in the middle disappears while two new clusters appear in the far right.

When using the extent as a measure of the quality of compression, the data bubbles (enclosed in a circle in part c of figure 7) that compressed the deleted cluster are eventually repositioned to another location. However, the insertion of the new clusters does not attract new data bubbles since they appear in a region where a previous data bubble is located (the enclosed data bubble in part d), which now summarizes more than one cluster after the insertions and deletions.

On the other hand, when using the fraction of points as the quality measure, the data bubbles are able to adapt to both the deletions and the insertions of clusters. The extent quality measure attempts to partition the space into roughly equal regions without regard to the point density. When a cluster is deleted, the data bubbles that compressed this cluster become empty and their extents are very small compared to the average extent. Thus, they are repositioned to new locations in the space. However, when new points possibly representing several sub-clusters are inserted, a close by data bubble can easily absorb all the sub-clusters without a significant change in its extent and its low compression quality is undetected by the extent measure. This data bubble now compresses significantly more points and the quality measure using the fraction of points instead of the extent identifies it as having a low compression quality whereby more data bubbles are repositioned to its vicinity, and the two new clusters are now compressed by several data bubbles instead of one (the

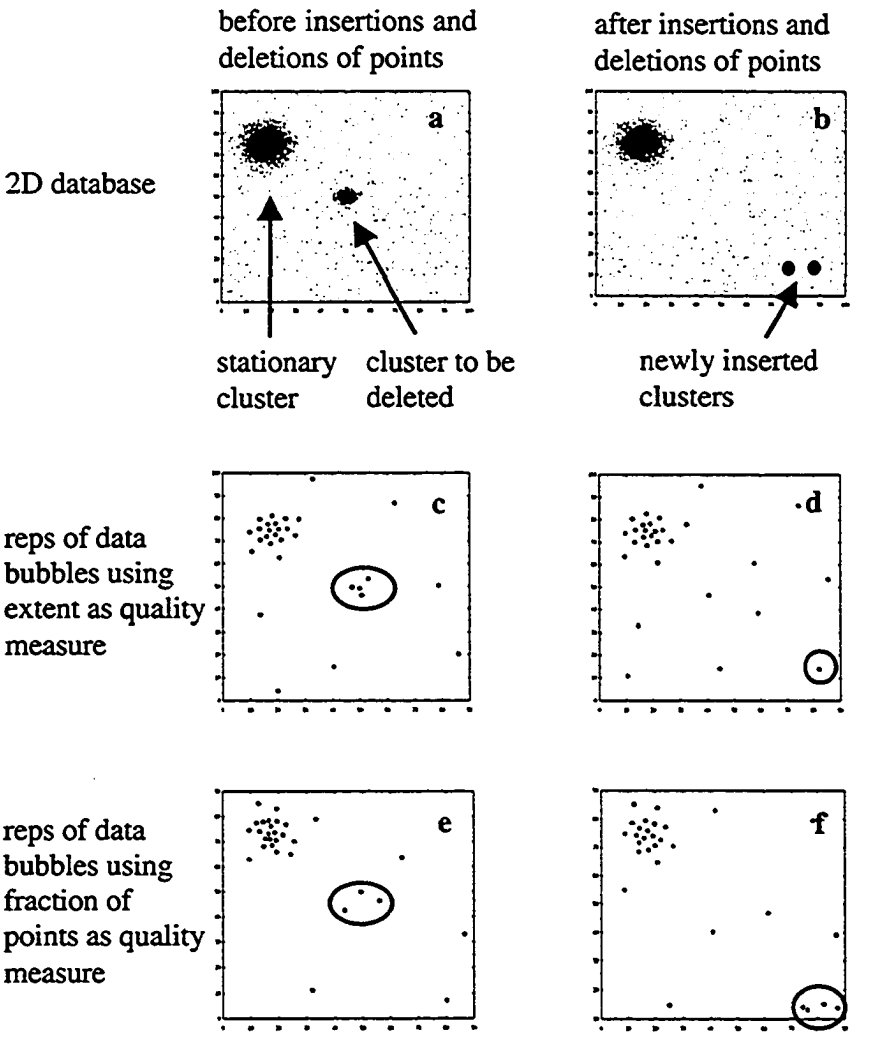

Figure 7. Comparison of Adaptation of Data Bubbles to Insertions and Deletions When Using the Fraction of Points vs the Extent as Quality Measures.

data bubbles identified in a circle in part f in figure 7). Thus, using the fraction of points is a much better quality measure than the extent to adapt data bubbles to the dynamic changes in a database.

Next we evaluate the performance of the incremental data bubbles using several databases. The performance of the incremental scheme is measured under the following dynamic situations of the database:

- Random: a database where points are inserted and deleted randomly according to the data distributions.
- Appear: a database where points are inserted and deleted such that a new cluster appears in the database over time.
- Extreme appear: a database where points are inserted and deleted such that a new cluster appears in the database over time but in a completely new region that does not contain any previous points, not even noise.
- Disappear: a database where points are inserted and deleted such that an old cluster disappears from the database over time.
- Gradmove: a database where one cluster gradually moves across the space over time via insertions and deletions.
- Complex: a combination of the above cases where there are random insertions and deletions to some clusters in the database, while other various clusters appear, disappear, and move with insertions and deletions of points as shown in figure 8.

We create databases using synthetic data to simulate the various scenarios described above which allow us to analyze the effectiveness of our scheme for different changes to the data distribution. We populate our databases with 50,000 to 110,000 points electing to simulate a reasonable average of the database size (smaller databases are easier to summarize while larger databases would yield similar results using proportionally more data bubbles for achieving the summarization).

Currently, we focus on achieving an effective data summarization capable of handling the dynamics of a given database with a certain percentage of insertions and

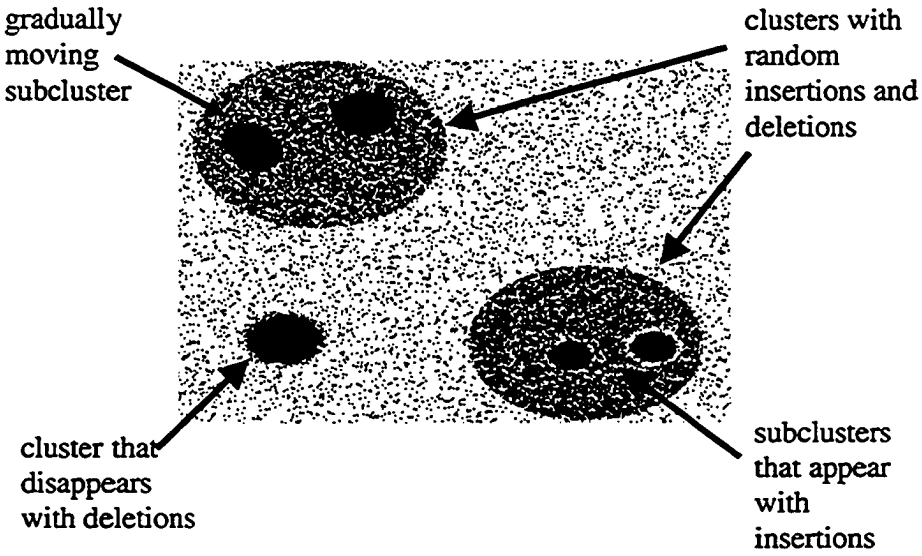

Figure 8. Clustering Structure in the Complex Database

deletions. In our databases, we assume that on average there will be an equal number of insertions and deletions (consistently inserting (or deleting) more points over time would cause the database to grow infinitely (or to disappear completely)). The probability needed to determine the boundaries of the classes of the data bubbles (presented above) was set to 80%. Using Chebyshev inequality, a probability of 80% results in a k value of 2. We used 200 data bubbles during the summarization of the databases, which is about 0.2% of size of the various databases (for analysis of different compression rates see [5]). We created databases with the above properties for several dimensions (2, 5, 10, 20). All results are average values of 10 repetitions of simulating the insertions and deletions.

We measure the quality and effectiveness of the incremental data bubbles by studying their effect on the performance of a clustering algorithm relative to its performance when using completely rebuilt data bubbles. After each batch of

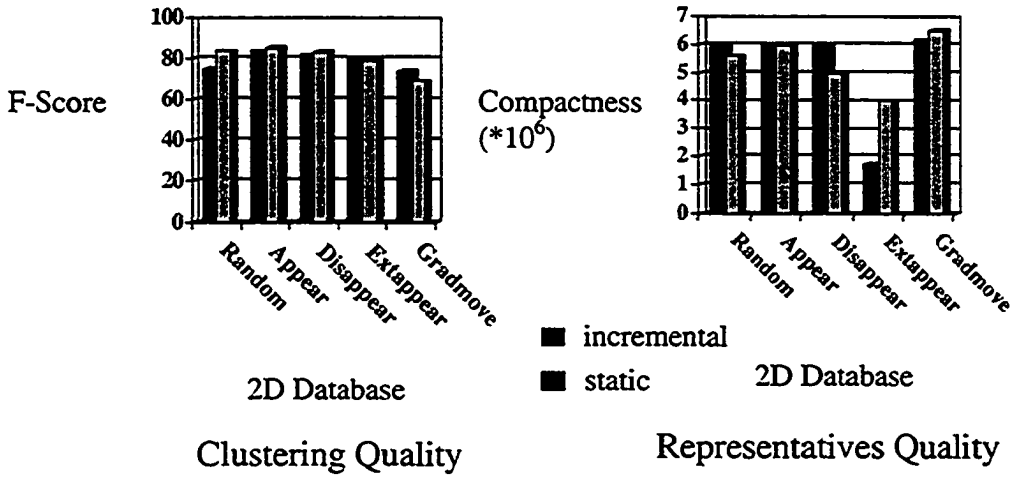

Figure 9. Incremental Scheme Quality using 2D Databases

insertions and deletions, we summarize each data base of the current points by building separate incremental and completely rebuilt data bubbles. Next, OPTICS is applied to these data bubbles separately to generate the reachability plots of the completely rebuilt and incremental clustering structures. The clusters are extracted from these plots using a modified version of an automatic method developed in [20]. The performance of OPTICS is determined using the F score measure [13] (where F = 2*p*r/(p+r), p is precision and r is recall).

We notice from figures 9 and 10 that the F score of the clustering algorithm OPTICS using our incremental scheme is always very close to (and sometimes higher than) the F score when using completely rebuilt data bubbles even when clustering the complex database. Thus, our scheme for maintaining the incremental data bubbles is effective in preserving both the quality of the data summarization and the quality of the clustering algorithm as measured by the F score.

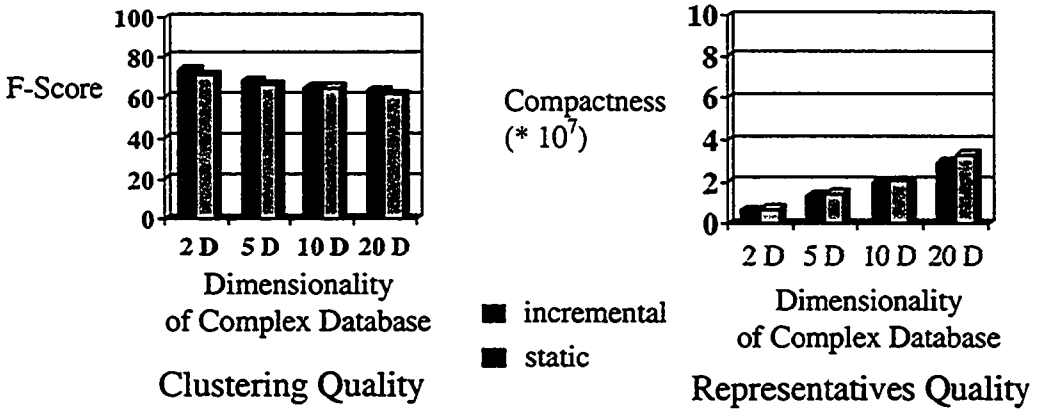

Figure 10. Incremental Scheme Quality using Complex Databases

To further analyze how our scheme of incremental data bubbles affects the quality of the data summarization technique, we study the effectiveness of repositioning the representatives of the rebuilt data bubbles in the proximity of the data points following a certain number of insertions and deletions. When a representative is close to its points, the compactness (which is the sum of the square distances of the points in the data bubble to its representative) is relatively low. In the completely rebuilt data bubbles, the representatives will be close to their points, with some possible variation in the positions of these representatives due to their random selection. If the repositioning of the representatives of incremental data bubbles is effective, then the overall compactness of the incremental data bubbles should not (significantly) exceed the overall compactness of the completely rebuilt data bubbles. As shown in figures 9 and 10, our dynamic scheme is very effective in (re)-positioning data bubbles. Incremental data bubbles even have a lower compactness than the completely rebuilt ones in many experiments. This effective (re)-positioning is further supported and reflected by the good clustering qualities (as indicated by the F scores) that we achieve.

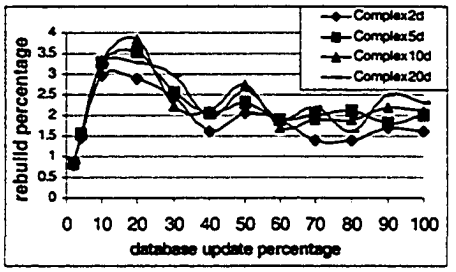

Figure 11. Percentage of Data Bubbles Rebuilt

We further analyze the adaptation of the incremental data bubbles to the dynamic updates to a database by examining the number of data bubbles that are rebuilt by our incremental scheme. Figure 11 shows the number of rebuilt data bubbles when applying our scheme to the dynamics of the complex database. On average, we rebuild only between 2 and 4 percent of the current data bubbles. This complex database contains various cases of dynamic changes to the clustering structures (a cluster disappears and reappears in completely different region of the space). Even in such highly dynamic cases, we never have to execute a rebuilding of all the data bubbles. Thus, the majority of the data bubbles are capable of adapting to changes in the data distributions by simply updating their sufficient statistics.

Furthermore, we study the effect of using the triangle inequalities in speeding up the assignment of points to data bubbles by measuring the number of distance calculations saved when utilizing the triangle inequalities (the overhead of computing the pair-wise distances among the representatives to utilize the triangle inequalities is low because typically the number of the representatives relative to the size of the database is small). Figure 12 shows the gain of using the triangle inequality in terms of the percentage of pruned distance computations when summarizing the complex

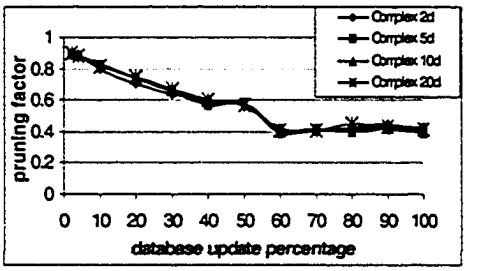

Figure 12. Pruning of Representatives using Triangle Inequalities

database. Typically, we can prune between 60 and 80 percent of all the distance computations using the triangle inequalities. This leads to significant gain in performance. This observation also indicates that a similar strategy is likely effective in significantly speeding up other techniques based on point assignment since those methods also basically execute distance computations.

In addition, we notice in figure 12 that the pruning factor decreases slowly as the fraction of updates in the complex database increases. As large amounts of points are inserted and deleted, the changes in the clustering structure in the complex database occur more abruptly, i.e. the clusters disappear and appear in larger batches. For instance, for the appear cluster, there are no initial representatives that are close to the points of the inserted cluster and can be used in the pruning. Only after the first batch has been inserted will there be close by representatives that can be used in the pruning. For the smaller fraction of insertions, the new region attracts a representative much earlier such that for points in later insertions the probability of avoiding distance computations to far away data bubbles is significantly higher.

CHAPTER FOUR

SPACE BUDGETED MAINTENANCE OF SUMMARIES FROM MULTI-DIMENSIONAL DATA STREAMS FOR EFFECTIVE CLUSTER ANALYSIS

4.1 Motivation

Knowledge discovery from data streams has attracted the interest of many scientists, as data streams are quickly becoming a prominent computational model and emerging in many planetary and extraterrestrial fields including astrophysics, financial markets, and phone and web services. Data streams pose numerous exciting computational challenges ranging from compact summarization of the stream, to effective pattern discovery, to efficient query processing. The ability of mining algorithm to solve these problems and provide accurate and efficient answers to stream queries in a flexible manner that is suitable for diversified applications is highly desirable.

A data stream is an instance of a block evolution model, where a data set is updated periodically through insertions and deletions [11]. In this model, the data set consists of conceptually infinite sequence of data blocks D_1, D_2, ... that arrive at times 1, 2, ... where each block has a set of records. Some applications require mining all of the data encountered thus far (unrestricted window scenario), while others require mining only the most recent part (restricted window scenario). There are several conditions on the representation of data streams [19]. First, a large portion of data arrives continuously and it is unnecessary or impractical to store all of the data. Second, the data points can be accessed only in the order of their arrival. Third, the

data arrives in chunks that fit into main memory. Moreover, there are also several requirements for the mining of data streams [3]. These requirements include: 1) compact representation of the points that can be maintained in main memory even as lots of new points arrive, 2) fast incremental processing of new data points, and 3) clear and fast identification of outliers. The interplay among these conditions and requirements, and their concurrent satisfaction makes knowledge discovery from data stream very interesting and challenging.

Cluster analysis from the distant past of a data stream hinges on effective summarization of the data stream. The summarization must not only be very space-efficient but also capable of efficiently providing high quality answers to stream queries. The problem of effectively and efficiently answering historical queries from data streams can be divided into two main sub-problems:

1. Effective and efficient construction and maintenance of a history of summarization of the data stream
2. Effective and efficient knowledge discovery from the summarization history

The formulation of the two main sub-problems and the performance of their solutions are interdependent. High quality (or effective and efficient) maintenance of the stream summarization is a precondition for high quality querying of the data stream, and high quality querying is a guiding condition for performing subsequent high quality maintenance and updating of the stream summarization. However, the solution to the problem of effectively and efficiently answering historical queries begins with effective and efficient historical summarization of the data stream.

In this chapter, we formulate and solve the first problem of efficiently constructing and maintaining a high quality and effective historical summarization of a multi-dimensional data stream. We present the main challenge in effectively summarizing a multi-dimensional data stream. In addition, we design various policies for maintaining both the summarization quality of the history and its overall space budget constraint.

The contributions of this chapter are:

1. A general framework for the summarization of multi-dimensional data streams.
2. Novel approximate spatio temporal summary suitable to effectively approximate several 'similar' spatial summary objects.
3. Dynamic stream summary stores that satisfy a given storage space budget constraint and provide effective answers to queries of historical stream windows.

4.2 Related Work

There has been a burst of research on data streams recently, with many efforts dedicated towards designing methods for compressing data streams and providing approximate answers to queries, and mining data streams. In this section, we quickly overview some of the related work relevant to summarizing data streams that contain dynamic clusters. As we discussed in the introduction, D. Barbara [3] and Ganti et al [11] presented the conditions and the requirements for modelling and mining data streams respectively. Several researchers have extended current database mining schemes to data streams mining, while adapting classical mining approaches.

Aggarwal et al [1] recently presented a framework for clustering evolving data streams by combining online micro clustering with offline macro clustering using BIRCH [27] to compress stream windows into clustering features. During the micro clustering phase, a temporal version of the clustering features of BIRCH and pyramidal time frames are used to store on disk micro clusters from different time snapshots in a pyramidal pattern. Once the user specifies the window for mining the macro clusters, the micro clusters for that window are extracted using the additivity property of the clustering features and the macro clusters are uncovered using a modified *k*-means algorithm that regards the micro clusters as points.

O'Callaghan et al [19] presented a new *k*-median algorithm called LocalSearch to solve a *k*-median problem that minimizes the facility cost function, where the cost associated with each cluster is estimated by considering the sum of the square distance of the points to the centres of the clusters. The Stream algorithm is presented to cluster each chunk of the stream using the LocalSearch algorithm.

In addition, researchers have also noted the application of signal processing techniques to compress streaming data and provide approximate answers to queries. Wavelets are a mathematical tool that transforms a signal into a set of so-called coefficients, where the first coefficient is the overall average of the signal and the remaining coefficients represent the overall 'shape' of the signal from coarse to fine respectively. By retaining all coefficients, the set can be inversely transformed to reconstruct the original signal, such that no information in S is lost. When several of the coefficients have small values, the space consumption of S can be reduced by

deleting these coefficients while introducing relatively small error when reconstructing the signal *S*.

To illustrate how wavelets work, we borrow an example from [26]. Assume we are given a signal S = [2,2,0,2,3,5,4,4] and we want to find the pattern in the signal. Using wavelets, we build this pattern hierarchically. In each step, we build the pairwise average of the current 'values' of the signal until one value is obtained which is the overall average of the signal. In the first step of transforming S using wavelets, we transform S to the following values [2,1,4,4], where the average of the first two values in S, (2,2), is 2, the average of the second two values of S, (0,2), is 1, and so on. To be able to inverse the values back to S, we save additional information by storing coefficients. *Haar* wavelets maintain as coefficients the pairwise differences of the original values. Thus, in the above example, we also store four coefficients: 0, -1, -1, 0, where (2–2)/2 = 0, (0-2)/2 = -1, and so on. We repeat this process recursively, until we obtain the overall average of the signal and the final set of coefficients as presented in table 1, where S is now transformed to $S^{*} = [2\frac{3}{4}, -1\frac{1}{4}, \frac{1}{2}, 0, 0, -1, -1, 0]$.

Table 1. Illustration of Wavelet Transformation

Resolution	Averages	Coefficients
8	[2,2,0,2,3,5,4,4]	
4	[2,1,4,4]	[0,-1,-1,0]
2	[1½,4]	[½,0]
1	[2¾]	[−1¼]

Chakrabarti et al [6] used high dimensional wavelets to build approximate synopses (wavelet coefficient synopses) of data streams such that queries can be approximately answered very quickly and entirely in the wavelets domain using special query processing algebra that is applicable to wavelet coefficients. Gilbert el al [12] also apply wavelets to stream data by building 'sketches' of the data that are suitable for approximately answering point and aggregate queries of data streams. A data stream is modelled as an incoming signal which is transformed using *Haar* wavelets and compressed by storing only a small number of the coefficients. When new data arrives, these coefficients are carefully maintained to preserve the quality of their approximation. Guha et al [13] also utilize wavelets to compress data streams and provide approximate query answers. Their approach, however, focuses on compressing multiple measures using so called extended wavelets, where an extended wavelet attempts to save storage space by storing the same wavelet coefficient present in different measures using a bit map. They provide schemes for maintaining these extended wavelets in a streaming fashion using a limited amount of space.

However, wavelets are not suitable for approximating clusters of high dimensional data, and, more importantly, their role in providing approximate answers to historically distant queries pertaining to evolutionary patterns of clusters present in dynamic data streams has not been envisioned yet. In this chapter, we overcome the high dimensionality limitation by first using an effective data summarization method suitable for compressing high dimensional data, and then utilize wavelets to further reduce the space consumption of the stream summarizations so that queries pertaining

to distant past regions of a multi-dimensional data stream can be effectively answered.

4.3 A General Framework for the Summarization of Multi-Dimensional Data Streams

In previous work, different types of data summarizations for multi-dimensional point data such as Clustering Features [27] and Data Bubbles [5] have been proposed to summarize static data sets. In addition, Clustering Features have been used to compress data streams [1], and Data Bubbles have been improved and utilized for compressing dynamic databases[18]. Both Clustering Features and Data Bubbles summarize in a sense sub-regions of the multi-dimensional space where data is located. In the following, we denote any of the summaries generated by these methods as *spatial summary objects.*

A data stream consists of an infinite sequence of data blocks –also called "windows"– W_1, W_2, ... that arrive at times 1, 2, ... A window-based data stream summarization can be consequently modeled as summarizing a (sub-) sequence of windows of the stream. In a limited amount of storage space, we can obviously only maintain a limited number of summary objects. A window based summarization of a data stream is defined as following.

Definition 4. *Given a data stream* $R=\{W_1, W_2, ...\}$ *of data point windows arriving at times 1, 2, ..., a (finite)* ***window-based summarization of R*** *is conceptually a sequence* $< SS^{W_{i_1}},...,SS^{W_{i_m}} >$ *where each element* $SS^{W_{i_j}}$ *is a summarization of a window* W_{i_j} *(at time stamp* i_j*) comprised of a number of spatial summary objects, i.e.:*

- *For each* $j \in \{1, ..., m\}$, *there is a* $k \geq 1$ *such that* $SS^{W_{i_j}} = \{S_1, ..., S_k\}$ *is a set of k spatial summary objects that summarize the points in a stream window* W_{i_j}, *where* $i_j \in \{1, 2, ...\}$.
- *If* $r < s$, *then* $SS^{W_{i_r}}$ *summarizes an earlier window than* $SS^{W_{i_s}}$. □

Clearly, given a window-based summarization $WS_R = < SS^{W_{i_1}}, ..., SS^{W_{i_m}} >$ of a data stream R, the sequence will maintain information about windows in a certain time interval $[t_S, t_E]$ of the stream, where the start time stamp $t_S = i_1$ and the end time stamp $t_E = i_m$.

The challenge for a data stream summarization is to describe a portion of the stream as large as possible using a window-based summarization in a finite amount of space such that important information about the data distribution in the stream can be re-constructed (e.g., using clustering algorithms). In practice, we want to be able to answer queries about portions of a stream. For that, we want to retrieve the information about all windows of the stream that belong to a query time interval. Since past stream windows are no longer available at query time, we can only answer the query by information retrieved from a stream summarization. In order to provide high-quality answers to queries pertaining to the distant past of a stream, the summarization should effectively manage its finite storage space.

A window-based summarization WS_R can be constructed (and maintained) using different methods. A naïve method is to keep just the m most recent window summarizations. Assuming that the available storage space can hold m window summarizations, once the space is filled up, a new window summarization is added

after deleting the oldest window summarization. This method obviously offers only a very limited stream time horizon and may store highly redundant information without considering any repetitive patterns in the stream.

The most recent proposal for summarizing multi-dimensional data streams in [1] can also be seen as an instance of our window-based summarization framework. This method improves over the naïve method by using pyramidal time frames to store summarizations of windows from different time stamps in a pyramidal fashion. The idea is to maintain a number of l so-called "time orders", where the i^{th} time order maintains a certain number of summarizations of windows with time stamps divisible by α^i for some user-specified value of α. This is conceptually equivalent to dividing the sequence $< SS^{W_1}, ..., SS^{W_m} >$ into l blocks (each of size m/l) to store the summarizations of windows belonging to different time orders. For instance, if α=2 (as used in the performance evaluation of [1]), the block for the first time order (2^0) will store the summarizations of the most recent m/l windows. The next block (time order 2^1) will store m/l summarizations of every second most recent window, and so on. For better space utilization, summarizations of windows that simultaneously belong to several time orders are only stored in the highest time order they belong to. The naïve method is still used for adding a new window summarization SS^W to its block: the oldest summarization in this block is deleted and SS^W is added.

The pyramidal time frames provide information within a larger time horizon than the naïve method. However, the available information is less accurate the more distant into the past it is. This scheme also does not consider any repetitive patterns in the stream to optimize storage utilization, and has the disadvantage that not only

different data streams may require different settings for the time orders (their number and the base of an order), but also the user has to somehow specify these parameters before initiating the summarization of the stream.

By further analyzing definition 4, we note that there are three main issues that have to be addressed when instantiating a window-based data stream summarization $< SS^{W_{i_1}} = \{S_1,..., S_{k_1}\},..., SS^{W_{i_m}} = \{S_1,..., S_{k_m}\} >$:

1. Determining which windows summarizations to keep in the sequence.
2. Determining which summary objects to keep in a certain window summarization.
3. Designing a space efficient encoding of the summary objects.

There are several thinkable policies for addressing each issue. To the best of our knowledge, the only proposal that has been presented for window based summarization of multi-dimensional data streams addresses only the first issue by keeping summarization windows according to a pyramidal time scheme [1].

We propose not to have a fixed policy for determining which window summarizations to keep based on a time stamp scheme but rather have the 'least informative' window summarizations removed. As we will see, we can achieve this goal by presenting effective policies for addressing the second and third issues.

To address the second issue, we propose two different policies for representing windows with different numbers of spatial summary objects. The first policy is age based where older window summarizations contain fewer summary objects. The second policy utilizes the "informativeness" of spatial summarizations objects based on its compression quality and the role it plays in the third issue.

To address the third issue, we propose two encoding schemes that effectively represent "similar" summaries by one "approximate" summary with a controlled error level and thus reducing the description length per summary object and more importantly enabling the applications of prominent compression techniques (Signal Compression and Clustering).

In our methodology, we use Data Bubbles as spatial summarization objects since our intended application is mining data streams of multi-dimensional hierarchical clusters, and Data Bubbles have been shown to significantly outperform other summarization methods for effectively uncovering hierarchical clusters. Our methodology is also applicable using other spatial summary objects such as Clustering Features.[2]

4.4 Approximating Spatial Summary Objects

4.4.1 (ε, δ) Approximate Spatial Configuration

In order to achieve effective window based summarization of a very large multi-dimensional data stream, we propose to encode several "similar" spatial summary objects that represent repetitive patterns in the underlying data distribution using one "approximate" summary object. The type of spatial summary objects we assume represents objects in a sub-region of the multi-dimensional space by sufficient statistics, which contains at least information equivalent to (*n*, *LS*, *SS*) as presented in section 4.3. This information is representing the data in a spatial region by the number of points, and some spatial information which is essentially the mean, i.e., the location, and the variance, i.e., the "spread", of the summarized data points. In order

[2] Although the information in a data bubble is more specialized than the basic sufficient statistics (*n*, *LS*, *SS*), it has been shown in [5] that the representative *rep*, the *extent*, and assuming a uniform distribution of points within a data bubble, the average nearest neighbour distances *nnDist(k,B)* can be easily derived from *n*, *LS*, *SS*.

to compare spatial summarization objects, we first define the spatial configuration of a spatial summary object as following.

Definition 5. *Given a spatial summary object S constructed from the clustering feature CF = (n, LS, SS), the* ***spatial configuration*** Ψ_S *of S is the tuple*

$$\Psi_S = (mean_S, \mathrm{var}_S)$$

where

- $mean_S = LS/n$

$\mathrm{var}_S = SS/n$ □

Using this definition, we can compare spatial configurations of spatial summarization objects S_1 and S_2 by comparing their variances and means. Intuitively, two summary objects are approximately equal if their representatives are 'close' and the two distributions they compress have similar variances. The idea is to check whether spatial summary objects from different windows have approximately equal spatial configurations such that their spatial configurations can be 'replaced' by one approximate spatial configuration object consuming less space. The information about the number of objects n in each of the spatial summarization objects is maintained separately enabling even further compression of the summary objects (as described in section 4.6). To define approximate equality of two spatial configurations, we simply require their means and variances to be within a certain error level of each other.

Definition 6. *Given* ε *and* $\delta \in [0,1]$*, and two summary objects* S_i *and* S_j*, then* S_i *and* S_j *have* ***(ε,δ)-approximately equal spatial configurations*** *iff the difference between their variances is within* ε *percent of the maximum of the two variances, and the distance*

between the two means is within δ percent of (the maximum of the) two standard deviations of the two spatial summary objects. Formally:

$$\Psi_{S_i} \approx_{(\varepsilon,\delta)} \Psi_{S_j} \Leftrightarrow$$

1. $\left|\mathrm{var}_{S_i} - \mathrm{var}_{S_j}\right| / \max(\mathrm{var}_{S_i}, \mathrm{var}_{S_j}) \leq \varepsilon$

2. $|mean_{S_i} - mean_{S_j}| / 2 * \max(\mathrm{stdev}_{S_i}, \mathrm{stdev}_{S_j}) \leq \delta$

where $mean_{Si}$ is the mean of points summarized by the summary object S_i, $stdev_{Si}$ is the standard deviation, and var_{Si} is the variance of points summarized by S_i □[3]

In an effective window based summarization of multi-dimensional data stream, the goal is to identify spatial summary objects $S_1,\ldots,S_k$, that belong to the summarizations of windows $i_1,\ldots,i_k$ respectively and each has a spatial configuration that is approximately equal to a certain spatial configuration, and therefore these summary objects represent approximately the same spatial pattern of points reoccurring in different windows. We define such an approximate spatial configuration for a set of spatial summary objects as following.

Definition 7. *Given a set of spatial summary objects $\hat{S}=\{S_1,\ldots,S_k\}$ with spatial configurations $\Psi_{S_1},\ldots,\Psi_{S_k}$ respectively, then an $\Psi^{\hat{S}}_{\varepsilon,\delta} = (m,v)$ is **(ε,δ)-approximate spatial configuration of the set $\hat{S}$** iff for all $S_i \in \hat{S}$: $\Psi_{S_i} \approx_{(\varepsilon,\delta)} \Psi^{\hat{S}}_{\varepsilon,\delta}$* □

[3] Note that the relation $\approx_{(\varepsilon,\delta)}$ is symmetric since we normalize by the maximum of both the mean and the standard deviation.

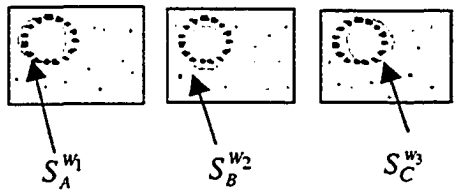

Figure 13. Summary Objects with Approximately Equal Spatial Configurations

If an (ε,δ)-approximate spatial configuration $\Psi_{\varepsilon,\delta}^{\hat{S}} = (m,v)$ of a set $\hat{S} = \{S_1,\ldots,S_k\}$ of spatial summary objects exists, then it holds that the spatial configuration of every spatial summary object is (ε,δ)-approximate equal to $\Psi_{\varepsilon,\delta}^{\hat{S}}$ (as defined in definition 6). By replacing the spatial configurations of each S_i with a reference to $\Psi_{\varepsilon,\delta}^{\hat{S}}$ we effectively reduce their space consumption while increasing the amount of storage space available for keeping more spatial summary objects that have different spatial configurations. The saved space is gained by loosing some accuracy in approximating the spatial configurations, where the accuracy loss is bounded by ε percent of the variance and δ percent of the mean of the approximated spatial configuration.

We test approximate equality only on spatial summary objects that are constructed from different stream windows. Spatial summary objects that are constructed from the same window are not compared because in practice, they do not satisfy the approximate equality condition by construction, as the summary objects typically divide the multi-dimensional space into large, disjoint sub-regions.

Figure 13. presents an illustration of the similarity of spatial summary objects. The figure shows spatial summary objects $S_A^{W_1}$, $S_B^{W_2}$, $S_C^{W_3}$ that are present in a region of the two dimensional space and are constructed from points in the data stream

windows W_1, W_2, and W_3. It is easy to see that, given small values for ε and δ, the spatial configurations of $S_A^{W_1}$, $S_B^{W_2}$, $S_C^{W_3}$ are (ε,δ)-approximately equal to the spatial configuration $\Psi_{\varepsilon,\delta}^{\{S_A^{W_1},S_B^{W_2},S_C^{W_3}\}}$ shown in the dotted circle in the figure.

4.4.2 Approximate Spatio-Temporal Summary Object

By replacing the spatial configurations of spatial summary objects $S=(n, LS, SS)$ in a set $\hat{S}$ with a reference to an (ε,δ)-approximate spatial configuration $\Psi_{\varepsilon,\delta}^{\hat{S}}=(m,v)$ we save the space needed to store the d-dimensional vector LS for each spatial summary object S_i, where instead of LS and SS, we only store a pointer (which consumes the same space as SS, which is just a number) to the approximate spatial configuration, which encodes an approximation of LS via $LS \approx n \cdot m$. To reconstruct the information of a spatial summary object from its approximation, we need to store the n value.

Two spatial summary objects whose spatial configurations are approximately equal may differ significantly only in the number of points n whereas their spatial configurations that encode their relative statistical properties are approximately equal. When these two summary objects represent points at different window time stamps, a change in the value of n from one window time stamp to the next represents an evolutionary pattern in the number of points present in the underlying distribution, where an increase represents emergence of points, a decrease represents dissolution of points, and no significant change represents stability of the distribution at the spatial location of these summary objects.

Consequently, if an (ε,δ)-approximate spatial configuration $\Psi^{\hat{S}}_{\varepsilon,\delta} = (m,v)$ of a set $\hat{S} = \{S_1,...,S_k\}$ of spatial summary objects exists, then the n values of the spatial summary objects can be viewed as a temporal sequence that conceptually represents a *life history* of the distribution D at the (approximate) location of the spatial summary objects. This life history consists of a sequence of time stamps $< t_1, t_2, ..., t_m >$ with an associated sequence of values $< n_1, n_2, ..., n_m >$, where n_k is the number of points in the summary object that summarizes points belonging to window with time stamp t_k, $1 \leq k \leq m$. The time stamps are increasing but not necessarily always consecutive, i.e., it is possible that some windows in the stream do not contain points at the location of the approximate spatial configuration $\Psi^{\hat{S}}_{\varepsilon,\delta} = (m,v)$. Conceptually, the sequence of time stamps (and its associated sequence of n values) can be considered as consisting of a chain of sub-sequences of consecutive time stamps $< t_i, t_i + 1, ..., t_i + k_i >$, i.e., in each subsequence the time stamps of adjacent pairs differ by one time unit.

We propose to store the sequences of time stamps and n values as a sequence of 'episodes', where each episode consists of start and end time points of a consecutive subsequence and stores the n values as a signal that can be separately compressed using signal compression techniques [23] or 'compared' to other signals such that 'similar' signals can be collectively compressed by replacing them with their common pattern using data mining techniques.

According to these considerations, we define a compact summary, called *approximate spatio-temporal summary*, that represents an approximate spatial configuration $\Psi^{\hat{S}}_{\varepsilon,\delta} = (m, v)$ and its associated temporal life history as following.

Definition 8. *Given an (ε,δ)-approximate spatial configuration $\Psi^{\hat{S}}_{\varepsilon,\delta}$ as defined in definition 7, and a sequence of increasing time stamps $t_{i_1}, t_{i_1}+1, \ldots, t_{i_1}+k_{i_1}, t_{i_2}, t_{i_2}+1, \ldots, t_{i_2}+k_{i_2}, \ldots, t_{i_m}, t_{i_m}+1, \ldots, t_{i_m}+k_{i_m}$ that are associated with a sequence of n values (of the spatial summary objects in a set $\hat{S}$) $n_{i_1}, n_{i_1+1}, \ldots, n_{i_1+k_{i_1}}, n_{i_2}, n_{i_2+1}, \ldots, n_{i_2+k_{i_2}}, \ldots, n_{i_m}, n_{i_m+1}, \ldots, n_{i_m+k_{i_m}}$ where $m \geq 1$, $k_{i_j} \geq 0$,*

an ***(ε,δ)-approximate spatio-temporal summary*** $\xi^{\hat{S}}_{\varepsilon,\delta}$ *is a tuple:*

$$\xi^{\hat{S}}_{\varepsilon,\delta} = (\Psi^{\hat{S}}_{\varepsilon,\delta}, < E^1, \ldots, E^m >)$$

where $E^j = (E^j_{start}, E^j_{end}, E^j_{signal})$, $1 \leq j \leq m$, *and* $E^j_{start} = t_{i_j}$, $E^j_{end} = t_{i_j} + k_{i_j}$,

$E^j_{signal} = < n_{i_j}, n_{i_j+1} \ldots, n_{i_j+k} >$ □

Approximate spatio-temporal summaries allow effective window based summarization of a multi-dimensional data stream such that historical queries pertaining to data points present in past temporal periods of the data stream can be answered with bounded error levels, and the evolution of spatial regions can be easily and effectively analyzed. Note that as a consequence of this definition, each spatial summary object in $\hat{S}$ must have a unique time stamp. That means an (ε,δ)-approximate spatio-temporal summary is allowed to summarize a sequence of spatial summary objects only from different stream windows.

In order to refer to the components of a spatio-temporal summary, we introduce the following notions.

Definition 9. *Given an (ε,δ) approximate spatio-temporal summary* $\xi^{\hat{S}}_{\varepsilon,\delta} = (\Psi^{\hat{S}}_{\varepsilon,\delta}, < E^1, ..., E^m >)$ *we call*

- $< E^1, ..., E^m >$ *the* ***Life-History*** *of* $\xi^{\hat{S}}_{\varepsilon,\delta}$
- E^i, $1 \leq i \leq m$ *an* ***Episode*** *of* $\xi^{\hat{S}}_{\varepsilon,\delta}$ □

To measure the "importance" of an approximate spatio-temporal summary $\xi^{\hat{S}}_{\varepsilon,\delta}$, we define the summarization error of $\xi^{\hat{S}}_{\varepsilon,\delta}$ as follows.

Definition 10. *Given an (ε,δ) approximate spatio-temporal summary* $\xi^{\hat{S}}_{\varepsilon,\delta}$ *that approximates n spatial summary objects, the* ***Summarization Error*** $SE(\xi^{\hat{S}}_{\varepsilon,\delta})$ *of* $\xi^{\hat{S}}_{\varepsilon,\delta}$ *is*

$$SE(\xi^{\hat{S}}_{\varepsilon,\delta}) = var^{\alpha}(\xi^{\hat{S}}_{\varepsilon,\delta}) / n^{\gamma}$$

where α, $\gamma \in \boldsymbol{R}$ *are coefficients for weighing the variance var of* $\xi^{\hat{S}}_{\varepsilon,\delta}$ *and n respectively.* □

Intuitively, an approximate spatio-temporal summary $\xi^{\hat{S}}_{\varepsilon,\delta}$ is more "important" than another spatio-temporal summary $\xi'^{\hat{S}}_{\varepsilon,\delta}$ when 1) $\xi^{\hat{S}}_{\varepsilon,\delta}$ represents more spatial summary objects than $\xi'^{\hat{S}}_{\varepsilon,\delta}$ and 2) its variance is smaller than the variance of $\xi'^{\hat{S}}_{\varepsilon,\delta}$. The coefficients α and γ add flexibility to the definition by allowing a user to give

different weights to the contribution of the variance and the contribution of n, respectively (in our experimental evaluation, we set $\alpha = \gamma = 1$).

4.5 Spatial Summarization of Data Streams

For the window based summarization of a data stream, it is important to design a scheme that not only efficiently summarizes the objects in a current window W_{i_c}, but also enables efficient finding of new spatial summary objects that have approximately equal spatial configurations to the spatial configurations of the previously constructed spatial summary objects such that approximate spatio temporal summaries can be quickly constructed and updated. The naïve approach for finding the best match for every spatial summary object from a set N in a set M is to compare all elements of the first set with all elements of the second set. The cost of this matching is $O(NM)$ comparison operations, which is computationally expensive. If the mean and the variance of the distribution of points compressed by a spatial summary object S_i do not change significantly from the current window to the next, then S_i remains "relatively" stationary in the data space and we can reuse the location of its representative to position a new spatial summary object S_j in the summarization of the next window, whereby the spatial configuration of S_j is only compared to the spatial configuration of S_i.

However, as noted in chapter 3, reusing locations of spatial summary objects that summarize a given data set D_1, to initialize spatial summary objects for a second data set D_2 that is expected to be similar to D_1, may result in so called "over-filled" spatial summary objects which summarize significantly more points than other summary objects, and have relatively low quality of summarization. We recall from definition 2

that the quality of a spatial summary object is measured using a data summarization index β. For an easy flow of presentation, we recall here definition 2 as follows.

Definition 11. *Given a data set D of N multi-dimensional points and a set $\hat{S}$ of spatial data summary objects that compress the points in D, the data summarization index β_i of a data summary object $S_i \in \hat{S}$ where S_i compresses n points is defined as* $\beta_i = \frac{n}{N}$ □

To identify summary objects that are "over-filled", i.e., have low quality of compression, definition 3 can be used (see chapter 3 for a more detailed and technical presentation). For an easy flow of presentation, we recall here definition 3 as follows.

Definition 12. *Given a data set D of N multi-dimensional points and a set $\hat{S}$ of spatial data summary objects that compress the points in D, let μ_β and σ_β be the mean and standard deviation of the distribution of the β values for all spatial data summary objects in $\hat{S}$. Given a value k (determined from a user-specified probability p according to Chebyshev's Inequality[24]), a spatial data summary object with the data summarization index β is called:*

1. *"good" iff* $\beta \in [\mu_\beta - k\sigma_\beta, \mu_\beta + k\sigma_\beta]$
2. *"under-filled" iff* $\beta < \mu_\beta - k\sigma_\beta$
3. *"over-filled" iff* $\beta > \mu_\beta + k\sigma_\beta$ □

In order to improve the overall quality of a summarization and eliminate over-filled spatial summary objects, the scheme presented in chapter 3 proposed to split each over-filled spatial summary object into two new spatial summary objects by re-positioning one spatial data summary objects that summarize relatively few points.

This strategy to maintain good quality spatial summary objects can also be applied in our current context of data streams. The summarization index β can be computed for summaries of data stream windows by considering each window as a data set. Splitting an over-filled summary object into only two new summary objects was suitable in the context of maintaining dynamic summaries of databases, where typically only very small difference between two large databases occurred. In a data stream, the differences between two consecutive windows can be more substantial so that a binary split may still generate overfilled spatial summary objects. To avoid this effect for stream windows, we propose to split an "over-filled" summary object simultaneously into $m = round(\beta_i / \mu_\beta)$ summary objects.

Our scheme for spatial summarization of stream windows is given as following.

To summarize the first window of a stream, spatial summary objects are constructed by selecting random points from the window as "seeds", and assigning each point to its closest seeds, incrementally computing sufficient statistics.

To summarize subsequent windows, spatial summary objects are constructed by re-using the spatial summary objects of the previous window. The means of the summary objects of the previous window are used as "seeds" for the current window, and all points in the current window are assigned to their closest seed, again incrementally computing sufficient statistics.

In all cases, after the initial construction of spatial summary objects, "over-filled" summary objects are identified using definition 12 and rebuilt as mentioned above.

4.6 Data Stream Summary Stores

We propose to encode a window based summarization WS_R of a multi-dimensional data stream R defined in definition 4 as a collection of (ε,δ) approximate spatio-temporal summaries as defined in definition 5. We call a collection of (ε,δ) approximate spatio-temporal summaries a *stream summary store*. Our general scheme for achieving an effective window based summarization using a summary store is presented as follows. We summarize each window in R into a set of spatial summary objects using the following general strategy.

- **Initialization of the Stream Summary Store**

1. Summarize the first window W_1 in the stream into a set of spatial summary objects $SS=\{S_1, \ldots, S_k\}$.
2. Initialize the summary store with a new spatio-temporal summaries $\xi_{\varepsilon,\delta}^{\{S_i\}} = (\Psi_{\varepsilon,\delta}^{\{S_i\}}, <E>)$ for each $S_i \in SS$, where $E = (E_{start}, E_{end}, E_{signal}) = (1,1,n_{S_i})$.

- **Maintenance of the Stream Summary Store**

1. Summarize a new window W_t, $t > 1$, in the stream into a set of spatial summaries $SS=\{S_1, \ldots, S_k\}$.
2. Integrate the new spatial summaries to the summary store:

   For every new spatial summary object S_i

   Search the summary store for a spatio-temporal summary $\xi_{\varepsilon,\delta}^{\hat{S}} = (\Psi_{\varepsilon,\delta}^{\hat{S}}, <E^1,\ldots,E^m>)$ with an (ε,δ)-approximate equal spatial configuration, i.e., $\Psi_{S_i} \approx \Psi_{\varepsilon,\delta}^{\hat{S}}$, and $E_{end}^m \neq t$

   If $\xi_{\varepsilon,\delta}^{\hat{S}}$ is found

update the life history of $\xi^{\hat{S}}_{\varepsilon,\delta}$ with n_{S_i} by either by extending the signal of the most recent episode E^m if $E^m_{end} = t-1$ or by adding a new episode $E^{m+1} = (t,t,n_{S_i})$.

else

construct a new spatio-temporal summary $\xi^{\{S_i\}}_{\varepsilon,\delta} = (\Psi^{\{S_i\}}_{\varepsilon,\delta}, < E >)$, where $E = (E_{start}, E_{end}, E_{signal}) = (t,t,n_{S_i})$ and add it to the summary store

3. Check the store's space budget and reduce space consumption by $m\%$ if needed:

   If compression of life histories of spatio-temporal summaries saves $m\%$ of the space consumed by the store

   compress life-histories

   else

   free $m\%$ of the space by deleting summaries with the largest summarization error SE as defined in definition 7

The details for integrating new spatial summary objects into a summary store and managing the store's space budget are discussed in the next two subsections.

4.6.1 Summary Store Life-History Management

To integrate a new spatial summary object S_i from a new window W_t into the summary store, we first have to check if there is a "matching" spatio-temporal summary. A matching spatio-temporal summary $\xi^{\hat{S}}_{\varepsilon,\delta} = (\Psi^{\hat{S}}_{\varepsilon,\delta}, < E^1, \ldots, E^m >)$ must satisfy the conditions that 1) it has an (ε,δ)-approximately equal spatial configuration to the

spatial configuration of S_i, and 2) $\xi^{\hat{S}}_{\varepsilon,\delta}$ has not yet integrated another spatial summary object S_j from the same window W_t, i.e., the $E^m_{end} \neq t$ (recall that a spatio-temporal summary is allowed to only summarize a sequence of spatial summary objects from different stream windows).

In general, to find a match for a spatial summary object S_i, we have to compare S_i to every spatio-temporal summary in the summary store until a matching summary is found. However, since we reuse locations of spatial summary objects of a previous window to position the seeds of spatial summary objects for a current window, we speed up the matching and improve its quality.

In general, the spatial configurations of many spatial summary objects (i.e., mean and the variance, but not their number of points) of the previous window are still suitable to represent the corresponding sub-regions in the current window. This is obviously true if the distribution does not change dramatically from one window to the next. But even if the distribution in the current window is overall very different from the previous window, on the "micro-clustering-level" of the spatial summary objects, the changes are not as drastic; the resulting spatial configurations of the current window are often within ε, δ (as defined in Definition 6) of the previous spatial configurations. As a consequence, we have a high probability of finding a matching spatio-temporal summary quickly by first checking the spatio-temporal summaries that have integrated the corresponding spatial summary objects. In addition, having good matches between consecutive windows leads in general also to fewer spatio-temporal summaries with longer episodes which results in a better space utilization. Note that constructing spatial summary objects for a current window

"from scratch" (e.g., using a random sample of the window as seeds) would not necessarily generate as many summaries that have approximately equal spatial configuration – even if the data distribution would be identical in the two windows.

4.6.2 Summary Store Space Management

Once the summary store is updated with new spatial summary objects of a new window, the space budget constraint of the summary store is enforced. We propose to reduce the space consumption of a summary store by 'compressing' the life histories of the approximate spatio-temporal summaries present in the summary store, and deleting some of these summaries when necessary. When we exceed the space budget after the integration of the summarization of the current window by *m%*, our scheme first tries to reduce the space consumption by compressing the life histories of spatio-temporal summaries. If this is not possible, *m%* of the space is released by discarding from the summary store the least 'important' summaries, which have the largest summarization error.

We propose to compress the life histories of approximate spatio-temporal summaries using one of two policies: local compression using signal compression, and global compression using clustering.

We can consider the sequence of n values in an episode of a spatio-temporal summary as a signal and model it using wavelets. We choose the orthonormal *Haar* transformation, which is very fast and suitable for data streams. When an episode is completed, i.e. ($E^m_{end} \neq t$) or the n signal reaches a certain user-specified maximum length, we normalize the signal using the *Haar* transform. Compression of a Wavelet transformed signal can be achieved by deleting the smallest coefficients from the

orthonormal basis which optimally minimizes the sum-squared error of the signal [23]. We note that to forward-transform/inverse-transform the signal when deleting coefficients, an episode must separately maintain the coefficients of the *n* signal and their coordinates.

In the local compression policy, *m%* of the space is saved as follows. For each episode, the number of coefficients that should be deleted to free *m%* of the space for the whole summary (including the space for start and end time points) is determined, and the smallest such coefficients are deleted. Episodes that do not have any remaining coefficients after the deletion are removed from their summary; spatio-temporal summaries that do not have any remaining episodes after this step are removed from the summary store. This procedure guarantees that at least *m%* of the total space is recovered. It is possible that more space is released when episodes or even whole summaries are completely deleted since an episode consumes space not only through its *n* signal but also its start and end time points, and a spatio-temporal summary also consumes space for its spatial configuration.

The global compression policy on the other hand globally saves *m%* of the space by approximating coefficients across multiple *n* signals instead of locally deleting coefficients from an *n* signal. Intuitively, to reduce the life history space by *m%*, the normalized *n* signals can be grouped into clusters. Each cluster is represented by an "average" *n* signal which approximates the elements in the cluster. Note that if two *n* signals have similar pair-wise changes in the *n* values, their sequences of coefficients are similar. Space compression is achieved by replacing the original *n* signal in an episode with a reference to the representative *n* signal of the assigned cluster. This

policy obviously only makes sense after "enough" episodes are constructed in the summary store so that the amount of space required for maintaining clusters of n signals is significantly smaller than the space required to maintain just the original n signals.

For clustering n signals of episodes, each *Haar* transformed signal can be considered a multi-dimensional point of signal values $p = (n_1^p, ..., n_d^p)$. One of the difficulties of clustering those n signals is that they may differ in their dimensionality or length d. If the dimensionality of points in a group G is not too different, a meaningful representative $r = (n_1^r, ..., n_{dmax}^r)$ can be computed as the "average" of all points in G, where the value of the i-th dimension is the average of the i-th dimension of all the points in G that have at least i dimensions, and $dmax$ is the largest length in G. Using this approach, when we want to use the representative as an approximation of a particular n signal $s = (n_1^s, ..., n_{d_s}^s)$ in G, $d_s \leq dmax$, we use only the first d_s dimensions of r, $\pi_{1,...,d_s}(r)$. The dissimilarity between two points $p = (n_1^p, ..., n_{d_1}^p)$ and $q = (n_1^q, ..., n_{d_2}^q)$ of different dimensionality can in this approach be defined as the Euclidean distance between the two points using the first number of coordinates present in both points, i.e., $dist(p,q) = \sqrt{\sum_{i=1}^{\min(d_1,d_2)} |n_i^p - n_i^q|^2}$. Consequently, to find meaningful representatives for groups of n signals such as an average n signal, we first have to partition the n signals into l "length-groups" where the signals in each partition have "approximately" the same length, and then find clusters and representatives within each partition. This can be achieved using different methods

such as an equal-width partitioning of the range of possible lengths into l segments (since we have a maximum length of episodes) or even using a clustering algorithm constructing l clusters.

When the global compression policy using clustering is applied for the first time, an initial clustering structure is constructed for every length-group of signals. Again, in principle any clustering algorithm that allows specifying the number of clusters k can be used with the above dissimilarity distance function for clustering the n signals in a group. The required number of clusters per group of n-signals has to be determined so that we achieve $m\%$ storage space reduction. Since we can compute how much space would be saved by replacing n-signals with a reference to a cluster representative, we can calculate the total number of clusters (over all groups) needed to achieve the required $m\%$ storage space reduction. The number of clusters per length-group is then allocated proportional to the number of signals in that group.

After the first application of the global compression policy, we have a set of clusters that have to be now maintained incrementally when episodes are deleted or added to the summary store. For incremental maintenance, the clusters are represented by the average n-signal of their members (as defined above) and the number of signals they contain. When the space consumption of new episodes must be reduced, the length-groups of their n signals are determined first, and then each signal is assigned to its closest cluster representative in this group, updating the average and incrementing the number of members in that group. When an episode is deleted, we know its cluster, and simply decrement the number of members in that cluster.

4.6.3 Reconstructing Summarized Stream Windows

We reconstruct the summarization set *SS* of a stream window W_i from a summary store that contains spatio temporal summaries in the following way. First, we find all spatio-temporal summaries $\xi^{\hat{S}}_{\varepsilon,\delta}$ whose life history contains an episode $E=(E_{start}, E_{end}, E_{signal})$ such that $E_{start} \leq i \leq E_{end}$, i.e., each $\xi^{\hat{S}}_{\varepsilon,\delta}$ approximates a spatial summary object S from window W_i, which is reconstructed by setting its spatial configuration to the spatial configuration of $\xi^{\hat{S}}_{\varepsilon,\delta}$. To set the n value of S, we recover from E the appropriate n value whose associated time point equals to i by first decompressing E_{signal} if E_{signal} was compressed earlier. On the one hand, the decompression of E_{signal} that was compressed using a local compression policy is achieved by first approximating the coefficients deleted during the compression by zeros and then inversing the approximate E_{signal} using the *Haar* wavelet. On the other hand, the decompression of E_{signal} that was compressed using a global compression policy is achieved by first constructing a temporary signal S' using the first m dimensions of the cluster E_{signal} is assigned to ($m = E_{end} - E_{start} + 1$), and then inversing S' using the *Haar* wavelet. Any $n \leq 0$ in an inverted signal is set to one.

4.7 Performance Evaluation

We perform extensive experimental evaluation of our stream summary store methodology. Our results demonstrate that our methodology is efficient and effective in several areas: summarization of stream windows, construction of a summary store of approximate spatio-temporal summaries in a limited space budget, integration of new spatial summary objects to the summary store, and maintenance of the summary

store to satisfy its space budget constraint. We test these various aspects of our methodology by querying the stores for historically distant information, and compare the retrieved information with the expected information obtained from the complete history. We query the stores for historically distant information by constructing from the store summarizations of the oldest windows in our streams. We consider queries that cover the summarizations of the first two, four, and eight windows of the stream. The complete history of the stream summary is constructed by archiving the complete spatial summary objects of all stream windows.

For a certain query, the relative quality Q measures the difference between the approximated spatial summaries $\hat{S}_{approx}$ obtained from the summary store to the exact spatial summaries $\hat{S}_{history}$ retrieved from the complete history. Intuitively, we want to measure how well $\hat{S}_{approx}$ approximates $\hat{S}_{history}$. For this, we first have to find for each spatial summary object in $\hat{S}_{history}$ the best matching spatial summary object in $\hat{S}_{approx}$. The quality of a matching between two spatial summary objects S_1, S_2, constructed from clustering features $CF_1 = (n_1, LS_1, SS_1)$ and $CF_2 = (n_2, LS_2, SS_2)$ respectively, is determined by how "close" they are in terms of their means, their number of points, and their variance. There is no trivial way to combine the changes in all three aspects into one single measure. We propose first to combine S_1 and S_2 by constructing the combined summary S_3, that represents both S_1 and S_2 using the additivity of clustering features [27], i.e., we construct $CF_3 = CF_1 + CF_2 = (n_3, LS_3, SS_3)$, with $n_3 = n_1 + n_2$, $LS_3 = LS_1 + LS_2$, and $SS_3 = SS_1 + SS_2$.

The difference in the sum squared error of S_3 and the sum of the sum squared errors of S_1 and S_2 relative to the sum squared error of S_3 tells us the amount of error

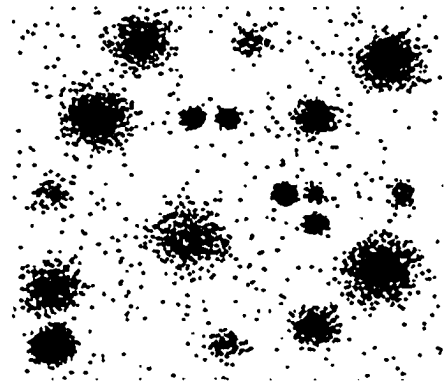

Figure 14. Snapshot of First Window in a Data Stream

we would introduce by substituting S_1 and S_2 by S_3. Formally, this relative mismatch of a fixed pair of spatial summary objects S_1 and S_2 is defined as $Mismatch(S_1, S_2) = (SSE_3 - (SSE_1 + SSE_2))/SSE_3$ where $SSE_i = \text{var}_i * n_i$. The smaller this value, the better S_3 represents the sum of S_1 and S_2, i.e., the better S_1 and S_2 "match". This mismatch measure is, however, monotonically dependent on the value of n_3, and therefore, to compare matches for a given spatial summary object S_1 with several other summary objects, we have to normalize by n_3. To compute the relative quality Q of the approximation $\hat{S}_{approx}$, we find the best matching spatial summary object in $\hat{S}_{history}$ for each spatial summary object in $\hat{S}_{approx}$. Note that the number of spatial summary objects in the two sets may be different due to possible deletions in $\hat{S}_{approx}$ during the maintenance of the summary store. The relative quality Q is computed as 1 minus the average mismatch between the spatial summary objects in $\hat{S}_{history}$ to the spatial summary objects in $\hat{S}_{approx}$ (in an optimal matching, the Mismatch for unmatched summary objects in $\hat{S}_{history}$ is counted as 1).

We evaluate the performance of our methodology using several multi-dimensional data streams, in which we simulate various dynamics of clusters starting with a spatial

configuration of the clusters in the first window of a stream that contains clusters as presented in figure 14. We set the size of a stream window to 10,000 objects. We simulate streams of 2, 5, and 10 dimensions, with four scenarios of evolution of clusters:

- **stream 1:** a stream consisting of 200 windows, where the means of the clusters are stationary but the clusters go through repeated periods of change in their variance (i.e., they "shrink" and "grow")
- **stream 2:** a stream consisting of 280 windows, where the means and variances of some clusters are stationary but for other clusters the number of points changes dramatically whereby they undergo repeated periods of disappearance, appearance, and no change (one cluster is deleted every window until all clusters are deleted, and then one cluster is added every window until all clusters are added, and so on)
- **stream 3:** a stream consisting of 140 windows, where the variance and the number of points in the clusters are stationary but their means change such that some clusters (two clusters) go through cyclic patterns of motion, some (five clusters) move randomly across the space, and others stay stationary
- **stream 4:** a stream consisting of 230 windows, which simulates a combination of the above three cases
- **stream 5:** a stream consisting of 300 windows, and which is not initialized with the configuration in figure 14, but constitutes a "worst case" scenario for data stream summarization, where each window contains independently of all

other windows a random number of randomly generated Gaussian clusters located at random positions in the data space.

We conduct our comparative analyses by examining the effectiveness of different window based summarization methods in answering queries pertaining to distant temporal regions in the stream. We query the stores for historically distant information by constructing from the store the summarizations of the oldest windows in our streams. We consider queries that cover the summarizations of the first two, four, and eight windows of the stream. To compare the effectiveness of the methods, we analyze at different time points the relative quality Q of the answers for any given query by repeatedly posing the same query after the arrival of new stream windows. As mentioned in section 3.3, Aggarwal et al [1] utilize time orders to save snapshots of the stream in a pyramidal time fashion. For comparison purposes, we assign our stream summary store a storage space budget that is equal to the sum of storage space budgets assigned to all orders. In our experiments, we set the number of orders to four and the base to 2 (i.e., the orders are 1, 2, 4, 8). We compare the performance of following four methods:

1. **Deletion**: maintain spatial summary objects in a pyramidal time fashion with deletion of oldest summary objects when storage space runs out (Aggarwal et al [1] method).
2. **Aggregation**: maintain spatial summary objects in a pyramidal time fashion with 'aggregation' of oldest summary objects when storage space runs out. Two summary objects S_1 and S_2 such that S_1 is constructed from the clustering feature $CF_1 = (n_1, LS_1, SS_1)$ and S_2 is constructed from the clustering feature

$CF_2 = (n_2, LS_2, SS_2)$, and summarize points from the same stream window, are 'aggregated' into one summary object S_3 constructed from $CF = (n_3 = n_1 + n_2, LS_3 = LS_1 + LS_2, SS_3 = SS_1 + SS_2)$ (when there is only one summary object left that summarizes a window W_i, it is simply deleted).

3. **Local Compression**: construct a stream summary store using our methodology and utilize the local compression policy to compress life histories when possible.

4. **Global Compression**: construct a stream summary store using our methodology and utilize the global compression policy to compress life histories when possible. We use the clustering algorithm Chameleon [15] to construct an initial signal clustering structure since it has been used previously to cluster wavelets [21], and therefore is suitable to cluster the transformed n signals (Chameleon is also used for partitioning the set of n signals into groups of similar lengths).

We set the number of spatial summary objects per window to 100, the probability for Chebyshev inequality to 0.8, the $m\%$ of space to be freed when space runs out to the fraction of space consumed by 100 spatial summary objects out of the total storage budget. The total storage budget is set to 16k, and ε and δ are set to 50%. By using a storage budget of 16k, we can analyze when the different methods run out of memory using our streams described above. Increasing/decreasing the storage space budget would only delay/hasten the exhaustion of the storage budget of the different methods and the comparative analysis of the different methods remains the same using streams with more/less windows where now the different methods exhaust their space budget after summarizing proportionally more/less windows. We compare the

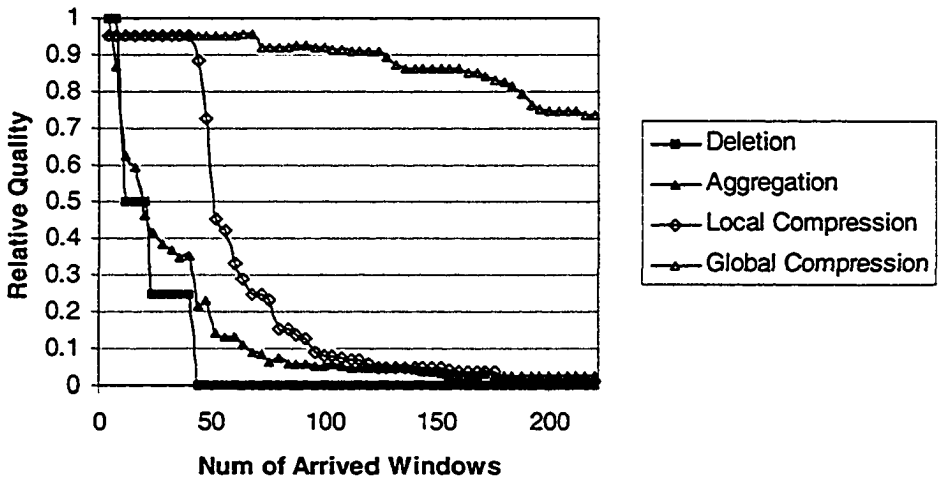

Figure 15. Relative Quality Profile When Querying 2D Stream 4 and Query Size is 4 Windows

accuracy profiles of the various schemes in answering at different time points during the simulation, the same query: retrieving the summaries in the first 4 windows of a stream. We use as time points the number of windows summarized since the beginning of stream summarization. In addition, we also examine the effect of changing the query size on the quality of the results, where we consider queries of sizes 2, 4, and 8 windows. As noted above, we currently use Chameleon. A few experiments run out of space during the clustering step in the global compression policy. Exploring other clustering algorithms is an endeavour for future directions.

Figure 15 shows the profile analysis when querying stream 4 that consists of 2 dimensional objects. Although the approximation of spatial configuration incurs a little loss in quality (as shown by a relative quality of less than 100% for the first few windows), the space gained by this approximation, and the life history compression techniques, allow our summary store to retain the majority of the required information and consequently the local and global compression methods outperform

the deletion and aggregation methods when querying the stream after summarizing the first 15 windows of the stream. However, as more windows are summarized, the local compression policy eventually deletes all the coefficients of the episodes that contain relevant information to answer the query and thereby the quality of the answer degrades dramatically at window 50. However, our summary store that utilizes the global compression policy (which uses clustering to compress life histories) drastically outperforms other schemes after window 50 as it effectively compresses the life history of episodes and is capable of providing answers with a high quality (about 70%) even after a very large number of the windows (200 windows) have been summarized.

To further demonstrate the effectiveness of our global compression policy in compressing the stream in comparison to other three methods (deletion, aggregation, and local compression), we conducted further experiments to determine how much storage space the other three methods need to maintain a quality of 70% after summarizing 200 windows of the stream. We found that the deletion and the aggregation methods required 15 folds more space than the storage space required by the global compression method, and that the local compression method required 4 folds more space than the storage space required by the global compression method.

The benefits from approximating spatial configurations are further evident when summarizing a data stream that consists of high dimensional objects. As shown in figures 16 and 17, the local and global compression methods provide high quality answers when increasing the dimensionality of the data space to 5 and then to 10.

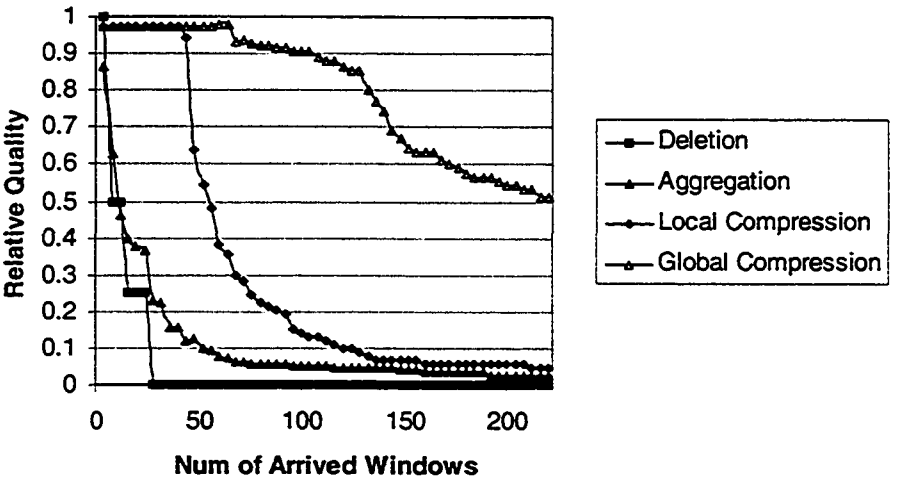

Figure 16. Relative Quality Profile When Querying 5D Stream 4 and Query Size is 4 Windows

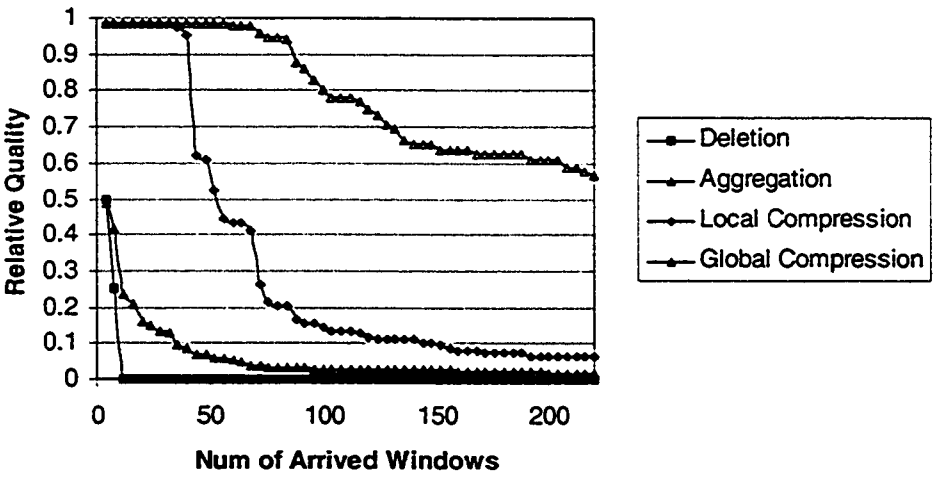

Figure 17. Relative Quality Profile When Querying 10D Stream 4 and Query Size is 4 Windows

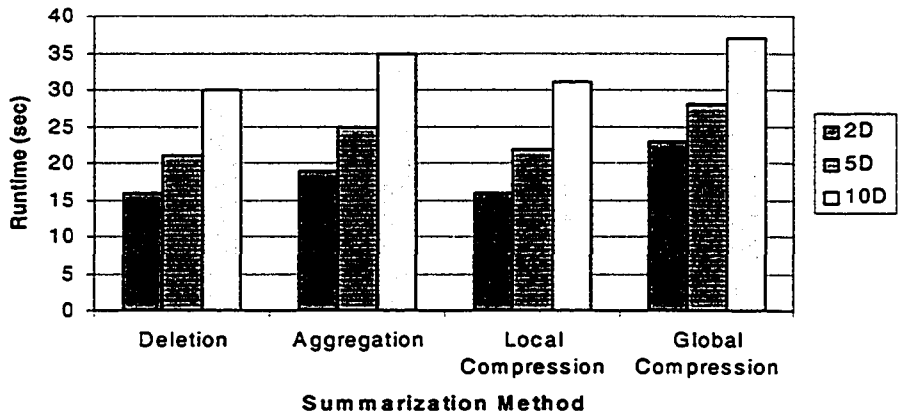

Figure 18. Summarization Method Runtime Comparison when Querying Stream 4 and Query Size is 4 Windows

However, the deletion and the aggregation methods (where there is no approximation of spatial configurations) spend a significant amount of the space storage budget on storing the high dimensional means without reducing this space storage by approximating them. Consequently, they exhaust their space storage budget very quickly and delete the summaries that contain the information for answering the query after summarizing only a few stream windows (when the dimensionality D is 10, the space budget has already been consumed by the fourth window, the time point when the first query arrives, and the quality of the query answer is only 50% of the quality of the query answer obtained from the archived summary history). At the same time, the runtimes of our local and global compression methods are very close to the runtimes of the deletion and aggregation methods, and consistently so across different dimensions as shown in figure 18.

Moreover, we compare the performance of the different methods when varying the query size from 4 windows to 2 and 8 windows. As figure 19 and 20 show, our

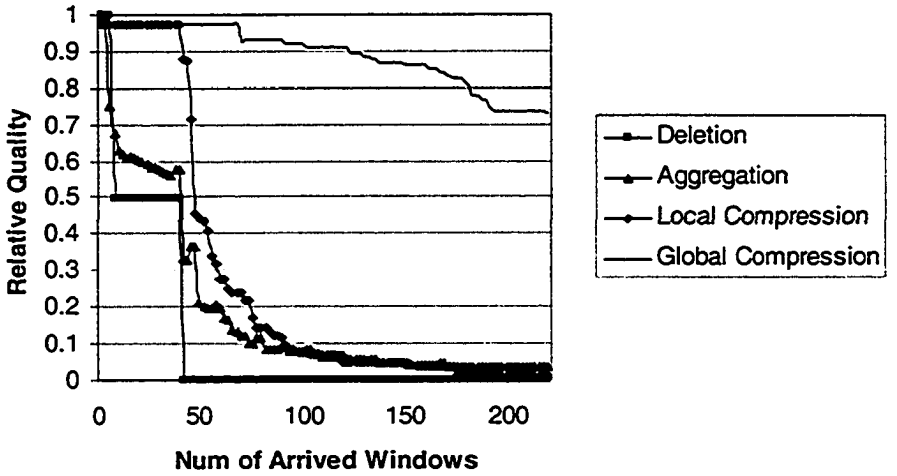

Figure 19. Relative Quality Profile When Querying 2D Stream 4 and Query Size is 2 Windows

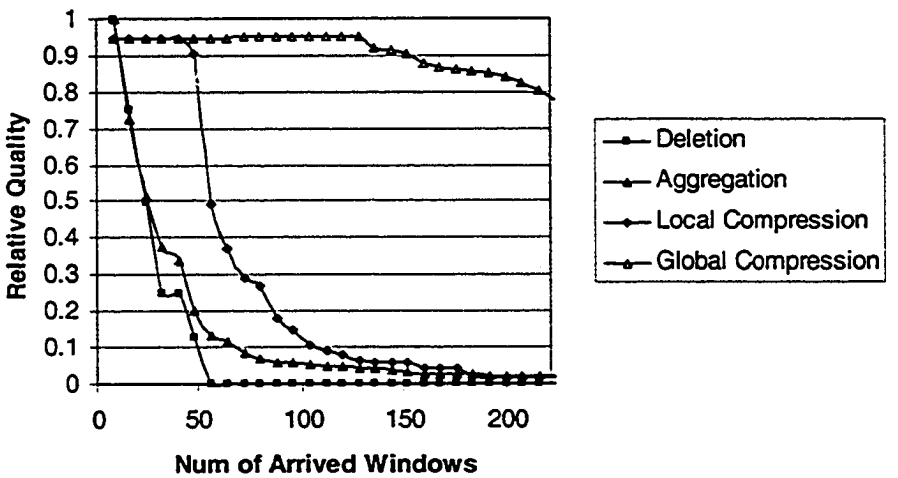

Figure 20. Relative Quality Profile When Querying 2D Stream 4 and Query Size is 8 Windows

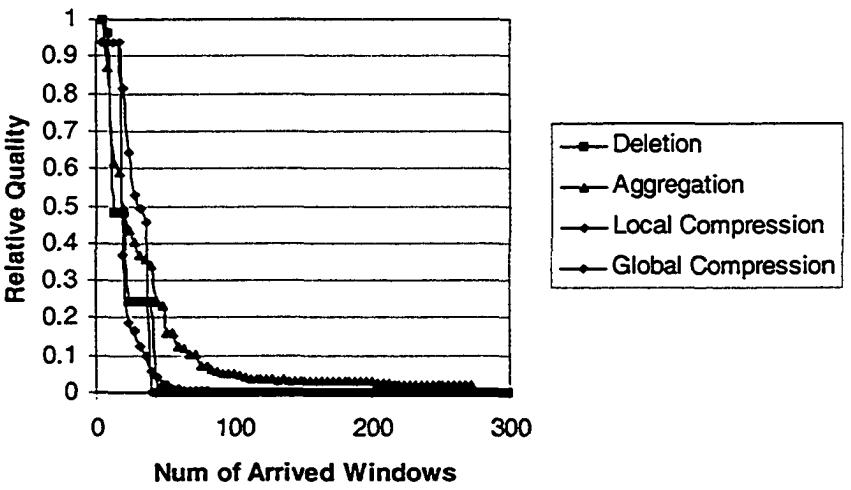

Figure 21. Relative Quality Profile When Querying 2D Stream 5 and Query Size is 4 Windows

method using global compression consistently and drastically outperforms other methods.

Using similar analysis, we examine the performance of the different methods when summarizing the extreme case of a stream 5 consisting of 2 dimensional objects. As figure 21 shows, all methods now provide answers with roughly the same quality (the results are similar across different dimensions of 2, 5, 10). Stream 5 consists of completely random changes to the clustering structure from one window to the next. Thereby, the majority of the spatial configurations of summary objects are completely different from one window to the next, and little space is saved by approximating the few similar ones. Thus, approximating spatial configurations of spatial summary objects is most effective when there are repetitive patterns in the data stream, and in that sense, our summary store has the added advantage that the number of approximate spatio temporal summaries it contains in comparison to the total number

of spatial summary objects constructed in the stream windows, provides some intuition about the existence of repetitive patterns in the stream.

CHAPTER FIVE

CONCLUSIONS

In this thesis, we presented a new scheme for incrementally maintaining effective data summarizations for the purpose of compressing large dynamic databases. Our incremental data bubbles are capable of handling various scenarios of insertions and deletions of points in a database environment and are suitable as an effective preprocessing technique for obtaining very efficient, online, hierarchical clustering analysis. A quality measure for the data bubbles was introduced to identify the data bubbles that do not compress well their underlying data points after certain insertions and deletions. We only rebuild these data bubbles using efficient split and merge operations. In addition, we also point out that data summarizations can be further sped up using triangle inequalities as illustrated by augmenting assignment of points to their closest data bubbles with triangle inequalities.

An extensive experimental evaluation for various cases of dynamic insertions and deletions of points in a database environment showed that the incremental data bubbles provide an efficient and effective data summarization technique of dynamically changing large databases, and even sometimes improve over data bubbles built from scratch while preserving the quality of the overall compression. Moreover, our scheme of incremental data summarization can augment further Data

Mining methods (like clustering techniques) to uncover hidden patterns in large databases very quickly.

Furthermore, we outlined a general window based summarization framework for compressing a multi-dimensional data stream. We presented a new methodology for achieving an effective window based summarization of a multi-dimensional data stream. Our comprehensive methodology effectively compressed similar spatial summaries from different window summarizations into one approximate spatio-temporal summary object, efficiently constructed a stream summary store of approximate spatio-temporal summaries, and effectively managed the integration of new window summarization into the summary store while satisfying the summary store space budget constraint by utilizing space management policies which we introduced and which used prominent Signal Compression and Clustering techniques.

An extensive experimental evaluation using a variety of data streams of different dimensions demonstrated that our methodology significantly outperformed existing summarization method. Our results demonstrated that our methodology is efficient and effective in several areas: summarization of stream windows, construction of a summary store of approximate spatio-temporal summaries in a limited space budget, integration of new spatial summary objects to the summary store, maintenance of the summary store to satisfy its space budget constraint. Moreover, unlike other methods, our stream summary store method is capable of providing high quality answers to queries that cover distant past regions of a multi-dimensional data stream while using a limited storage budget.

CHAPTER SIX

FUTURE DIRECTIONS

There are several interesting directions for future research of knowledge discovery in multi-dimensional data streams using our general framework of window based summarization and stream summary stores. First, formulating and solving the problem of cluster evolution analysis in a multi-dimensional data stream using the presented stream summary store summarization methodology is a very interesting computational problem to consider. What are the computational tools needed to model and uncover evolutionary patterns of clusters development in a multi-dimensional data stream? In addition, mining patterns from several stream summary stores to effectively compress several multi-dimensional data streams simultaneously is an intriguing problem. What is an effective strategy to summarize several multi-dimensional data streams that may belong to different applications? Is maintaining a summary store for each such stream a good method? Or is it possible to design a more effective method consisting of a 'general' summary store with different 'departments', where now the store abstracts and maintains similar 'patterns' from the departments such that the overall storage space requirement is significantly smaller than the total storage requirements of the separate stores?

Furthermore, in this thesis, we solved the first problem of achieving effective window based summarization of a multi-dimensional data stream. It is now also very interesting to look into the second complementary problem: online mining of a stream summary store. Is it possible that there are regions in a multi-dimensional data

streams that are queried less frequently such that a stream summary store can discard most of the information pertaining to these regions? Moreover, it is possible to formulate and find patterns in the queries of a stream summary store and utilize these patterns to provide "mining feedback" to the store about its space management?

Certainly, upcoming applications will call upon effective and intelligent techniques for knowledge discovery in multi-dimensional data streams. The envisioning of computational problems and providing effective solutions to them in this realm will be most gratifying.

CHAPTER SEVEN

BIBLIOGRAPHY

[1] Aggarwal, C. C., Han, J., Wang, J., Yu, P. S. A Framework for Clustering Evolving Data Streams. In VLDB'03, 81-92, 2003.

[2] Ankerst, M., Breuing, M., Kriegel, H-P., Sander, J. OPTICS: Ordering Points to Identify the Clustering Structure. In SIGMOD'99, 49-60, 1999.

[3] Barbara, D. Requirements for Clustering Data Streams. SIGKDD Explorations, 3:23-27, 2002.

[4] Braunmueller, B., Ester, M., Kriegel, H-P., Sander, J. Efficiently Supporting Multiple Similarity Queries for Mining in Metric Databases. In ICDE'00, 256-267, 2000.

[5] Breuing, M., Kriegel, H-P, Kroger, P., Sander, J. Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering. In SIGMOD'01, 79-90, 2001.

[6] Chakrabarti, K., Garofalakis, M., Rastogi, R., Shim, K. Approximate Query Processing Using Wavelets. In VLDB'00, 111-122, 2000.

[7] Chen, C., Hwang, S., Oyang, Y. An Incremental Hierarchical Data Clustering Algorithm Based on Gravity Theory. In PKDD'02, 237-250, 2002.

[8] Elkan, C. Using the Triangle Inequality to Accelerate *k*-means. In ICML'03, 147-153, 2003.

[9] Ester, M., Kriegel, H-P., Sander, J., Xu, X. A Density Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In KDD'96, 226-231, 1996.

[10] Ester, M., Kriegel, H-P., Sander, J. Wimmer, M., Xu, X. Incremental Clustering for Mining in a Data Warehousing Enviornment. In VLDB'98, 323-333, 1998.

[11] Ganti, V., Gehrke, J., Ramakrishnan, R. Mining Data Streams under Block Evolution. In ACM SIGKDD Explorations, 3:1-10, 2002.

[12] Gilbert, A., Kotidis, Y., Muthukrishnan, S., Strauss, M. J. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries. In VLDB'01, 79-88, 2001.

[13] Guha, S., Kim, C., Shim, K. XWAVE: Optimal and Approximate Extended Wavelets For Streaming Data. In VLDB'04, 288-299, 2004.

[14] Jawerth, B., Sweldens, W. An Overview of Wavelet Based Multiresolution Analyses. In SIAM Rev., 36(3): 377-412, 1994.

[15] Karypis, G. Han, E. H., Kumar, V. Chameleon: A Hierarchical Clustering Algorithm Using Dynamic Modeling, IEEE Computer, 32(8):68-75, 1999.

[16] Larsen, B., Aone, C. Fast and Effective Text Mining Using Linear-time Document Clustering. In KDD'99, 16-22, 1999.

[17] MacQueen, J. Some Methods for Classification and Analysis of Multivariate Observations. In 5th Berkeley Symp. Math. Statist. Prob., 281-297, 1967.

[18] Nassar, S., Sander, J., Cheng, C. Incremental and Effective Data Summarization for Dynamic Hierarchical Clustering. In SIGMOD'04, 467-478, 2004.

[19] O'Callaghan, L., Mishra, N., Meyerson, A., Guha, S., Motwani, R. Streaming-data Algorithms for High-quality Clustering. In ICDE'02, 685-704, 2002.

[20] Sander, J., Qin, X., Lu, Z., Niu, N, Kovarsky, A. Automated Extraction of Clusters from Hierarchical Clustering Representations. In PAKDD'03, 75-87, 2003.

[21] Sheikholeslami, G., Chatterjee, S., Zhang, A. WaveCluster: a Multi-Resolution Clustering Approach for Very Large Spatial Databases. In VLDB'99, 428-439, 1998.

[22] Sibson, R. SLINK: An Optimally Efficient Algorithm for the Single-link Cluster Method. The Computer Journal, 16(1): 30-34, 1973.

[23] Sollnitz, E. J., Derose, T. D., Salesin, D. H. Wavelets for Computer Graphics. Morgan Kaufmann, 1996.

[24] Triola, M. F., Goodman, W. M., Law, R. Elementary Statistics First Canadian Edition, Addison-Wesley, Don Mills, Ontario, 1999.

[25] Widyantoro, D. H., Ioerger, T. R., Yen, J. An Incremental Approach to Building a Cluster Hierarchy. In ICDM'02, 705-708, 2002.

[26] Yossi, M., Vitter, J. S., Wang, M. Dynamic Maintenance of Wavelet Based Histograms. In VLDB'00, 101-110, 2000.

[27] Zhang, T., Ramakrishnan, R., Linvy, M. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In SIGMOD'96, 103-114, 1996.