Incremental and Effective Data Summarization for Dynamic Hierarchical Clustering

Samer Nassar  
University of Alberta  
Department of Computing Science  
Edmonton, Alberta, Canada  
T6G 2E8  
samer@cs.ualberta.ca

Jörg Sander  
University of Alberta  
Department of Computing Science  
Edmonton, Alberta, Canada  
T6G 2E8  
joerg@cs.ualberta.ca

Corrine Cheng  
University of Alberta  
Department of Computing Science  
Edmonton, Alberta, Canada  
T6G 2E8  
corrine@cs.ualberta.ca

ABSTRACT

Mining informative patterns from very large, dynamically changing databases poses numerous interesting challenges. Data summarizations (e.g., data bubbles) have been proposed to compress very large static databases into representative points suitable for subsequent effective hierarchical cluster analysis. In many real world applications, however, the databases dynamically change due to frequent insertions and deletions, possibly changing the data distribution and clustering structure over time. Completely reapplying both the summarization and the clustering algorithm to detect the changes in the clustering structure and update the uncovered data patterns following such deletions and insertions is prohibitively expensive for large fast changing databases. In this paper, we propose a new scheme to maintain data bubbles incrementally. By using incremental data bubbles, a high-quality hierarchical clustering is quickly available at any point in time. In our scheme, a quality measure for incremental data bubbles is used to identify data bubbles that do not compress well their underlying data points after certain insertions and deletions. Only these data bubbles are re-built using efficient split and merge operations. An extensive experimental evaluation shows that the incremental data bubbles provide significantly faster data summarization than completely re-building the data bubbles after a certain number of insertions and deletions, and are effective in preserving (and in some cases even improving) the quality of the data summarization.

Keywords

Data Summarization, Incremental Data Bubbles, Clustering

1. INTRODUCTION

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 [14] create k partitions of the points. Hierarchical clustering algorithms like the Single-Link method [17] 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 [16].

In recent years and with the massive increase in the size of databases, 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 to larger databases 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 [20], 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. For example, OPTICS [2] was shown to uncover the clustering structure effectively and very efficiently from special data summaries that compress large databases [5].

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.

The problem of incrementally mining large dynamic databases is to some degree related to the problem of mining data streams. However, the requirements for the algorithms solving these problems are different in both scenarios. In the data stream model, it is assumed that the storage is extremely limited and the data coming in from a stream is not stored in a database. A data mining algorithm will see the data stream as a sequence of time ordered “windows” that contain a very limited number of objects at a given time.

In contrast, in an incremental database, the data is inserted and deleted over time due to some application logic (and not necessarily according to time stamps). The intended applications include not only data mining, but also other business tasks. The size of the database available to an application at any given point in time is typically very large and its dynamics can not in general be “simulated” for the data stream based algorithms, e.g., by a sliding window approach. A data stream, however, can be regarded to a certain degree as a degenerate case of an incremental database where the database size is extremely small (the size of a window in a stream), and insertions and deletions arise such that the current “database” content is completely replaced. We focus in this paper on achieving incremental summaries of dynamic databases.

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 paper, 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 to adapt 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 [20], 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 paper, 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 paper 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 for the database using only the data bubbles such as hierarchical clustering structures.

The paper is organized as follows. Section 2 presents a review of work related to the problems of incremental clustering and data summarization in the context of data streams. In section 3, we show how to speed up the construction of data bubbles (both incremental and static) by utilizing triangle inequalities during the assignment of points to their closest data bubbles. The scheme for incrementally maintaining data bubbles with good qualities of compression is presented in section 4. In section 5, we perform an
extensive experimental evaluation of our method, showing that incremental data bubbles are significantly faster to maintain than completely re-built data bubbles, while preserving (and in some cases even improving) the quality of the data summarization for hierarchical clustering. The conclusions and some future directions are presented in section 6.

2. RELATED WORK

The problem of incremental clustering has been studied by many scientists. We discuss some of the proposed algorithms, and overview the different requirements necessary for incremental clustering in the context of data streams. There are several incremental clustering algorithms that do not use the data summarization technique but attempt to directly restructure the clusters to reflect 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 sets. 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. 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.

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. 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. [19] 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. 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.

Charikar et al. [6] introduce new deterministic and randomized incremental clustering algorithms while trying to minimize the maximum diameters of the clusters. The diameter of a cluster is its maximum distance among its points and is used in the restructuring process of the clusters. When a new point arrives, it is either assigned to one of the current clusters or it initializes its own cluster while two existing clusters are combined into one.

As indicated in the introduction, the clustering of a data stream is to some degree related to the incremental clustering of dynamically changing databases, although data streams impose different requirements on the mining process and the clustering algorithms. The purpose of both methodologies is to provide the user with “up-to-date” clusters very quickly from dynamic data sets. However, when mining a dynamically changing database, the clustering algorithm has access to all points in the data base and not necessarily only the most recently inserted points, and the algorithm is not restricted to a sequential access to these new points. The clustering algorithm has slightly different requirements when mining data streams as explained next.

D. Barbara [4] outlines the main requirements for clustering data streams. These requirements consist of 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 cluster assignment of new points should use a function that does not depend on comparison to past points and yet performs well.

Ganti et al. [12] also examine mining of data streams. A block evolution model is introduced where a data set is updated periodically through insertions and deletions. In this model, the data set consists of conceptually infinite sequence of data blocks D1, D2, ... 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) and updating the models accordingly. The authors highlight two challenges in mining evolving blocks of data: change detection and data mining model maintenance. In change detection, the differences between two data blocks are determined. Next, a data mining model should be maintained under the insertions and deletions of blocks of the data according to a specified data span and block selection sequence.

The data stream model is also discussed by O’Callaghan et al. [15], who indicate that the model handles the following cases when mining dynamic data sets. 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. A new k-median algorithm called LocalSearch is presented 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 centers of the clusters. The Stream algorithm is presented to cluster each chunk of the stream using the LocalSearch algorithm.

Aggarwal et al. [1] use the data summarization method BIRCH [20] in the context of mining data streams. BIRCH compresses a dataset into so-called clustering features (CFs). A clustering feature \( CF = (n, LS, SS) \), where \( LS \) is the linear sum of the \( n \) points compressed by the CF and \( SS \) is their square sum. The scheme presented by Aggarwal et al. [1] for clustering data streams combines online micro clustering with offline macro clustering [1]. 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.

3. SPEEDING-UP THE CONSTRUCTION OF DATA BUBBLES

In this paper, we consider the problem of using incremental data summarizations to speed up incremental hierarchical clustering of large databases. 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 the points and \( SS \) is their square sum, as proposed, e.g., for BIRCH. 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:

**Definition 1.** A data bubble \( B \) for a set of points \( X = \{X_i\}, 1 \leq i \leq n \) is a tuple \( B = (\text{rep}, n, \text{extent}, \text{nnDist}) \) where

- \( \text{rep} \) is a representative, defined as the mean of the points in \( X \)
- \( n \) is the number of points in \( X \)
- \( \text{extent} \) is the radius of \( B \) around \( \text{rep} \) that encloses the majority of the points in \( X \)
- \( \text{nnDist}(k,B) \) 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 \( \text{rep} \), the \( \text{extent} \), and assuming a uniform distribution of points within a data bubble, the average nearest neighbor distances \( \text{nnDist}(k,B) \) can be easily derived from \( n, LS, SS \).

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. 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:

**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 \( \text{dist}(s_{B1}, s_{B2}) \geq 2 \text{dist}(p, s_{B1}) \), then \( \text{dist}(p, s_{B1}) \leq \text{dist}(p, s_{B2}) \). □

The lemma is illustrated in Figure 1. Assume we have precomputed 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 the lemma 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 \( s_i \) 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 \( s_j \) of all the other data bubbles without computing the overhead of computing distances among the seeds of all of the data bubbles. We try to avoid distance computations that can be easily derived from \( n, LS, SS \). The selected seeds of two data bubble \( s_{B1} \) 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 the lemma that \( s_{B2} \) cannot be closer to \( p \) than \( s_{B1} \).

**Figure 1.** Pruning distance calculations.

---

1 Related techniques for pruning distance calculations using triangle inequalities have been successfully applied in the computation of similarity queries [3], [11] and in k-means [8]
set CandidateSeeds to the set of all seeds of data bubbles
select and remove a random seed $s_i$ from CandidateSeeds
compute $\text{minDist} = \text{dist}(p, s_i)$
while CandidateSeeds is not empty
  for all $s_j$ in CandidateSeeds
    look up the distance $d_{s_j, s_c}$ between $s_j$ and $s_c$
    if $d_{s_j, s_c} >= 2 \times \text{minDist}$
      remove $s_j$ from CandidateSeeds
  while CandidateSeeds is not empty
    select and remove a random seed $s_j$
    compute $\text{dist}(p, s_j)$
    if $\text{dist}(p, s_j) < \text{minDist}$
      set $s_c = s_j$
      set $\text{minDist} = \text{dist}(p, s_j)$
    break
return $s_c$

Figure 2. Finding the closest data bubble seed for a point $p$.

their distances to $p$ by looking up the distances between $s_c$ and $s_j$
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 the 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.

4. INCREMENTAL DATA BUBBLES

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 $M\%$
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 $p$, 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.

4.1 The 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 [20], or its extension to data streams [1], 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 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 \( \beta \) that we define to be the fraction of points in the database compressed by the data bubble.

**Definition 2.** Given a database \( D \) of \( N \) points and a set \( \Omega \) of data bubbles that compress the points in \( D \), the data summarization index \( \beta_i \) 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 \( \beta \) 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 \( \beta \) values in order to determine which data bubbles have a good quality of compression and which do not.

In a set \( \Omega \) of data bubbles that compress the database \( D \), the \( \beta \) 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 \( \beta \) 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 \( \beta \) values, we can determine the outliers in the distribution by estimating the lower and upper boundaries of the \( \beta \) interval that characterizes the “good” data bubbles through using Chebyshev’s Inequality theorem [18]. According to the theorem, if \( \mu_x \) and \( \sigma_x \) are the mean and standard deviation of a random variable \( X \), then for any positive constant \( k \)

\[
P( X \leq \mu_x - k\sigma_x ) \leq 1 - \frac{1}{k^2}
\]

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 \( \beta \) as a random variable with the mean \( \mu_\beta \) and the standard deviation \( \sigma_\beta \) 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 \( \beta \) 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 \( \beta \) value would be significantly larger than the average, and therefore its \( \beta \) value
would be above the upper boundary $\mu_\beta + k\sigma_\beta$ (the $\beta$ value would be located towards the right end of the distribution). On the other hand, $\beta$ values that are significantly lower than the average $\beta$ value are below the lower boundary $\mu_\beta - k\sigma_\beta$. These low $\beta$ 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 $\Omega$ of data bubbles that compress the points in $D$, let $\mu_\beta$ and $\sigma_\beta$ be the mean and standard deviation of the distribution of the $\beta$ values for all data bubbles in $\Omega$. 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 $\beta$ 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 $\beta > \mu_\beta + k\sigma_\beta$

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.

### 4.2 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 reassigned some of these points among more data bubbles, whereby the old data bubble gives up some of its points to other data bubbles.

The naïve approach is to reassigned 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, reassigned 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.

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 $\beta$ values and contain relatively few (or no) points that can be distributed among neighboring data bubbles without significantly affecting the

---

![Figure 4. Over-filling of a data bubble by new clusters.](image-url)
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 $B_{\text{over-filled}}$ is improved by merging and ideally re-positioning an under-filled data bubble $B_{\text{under-filled}}$ to the vicinity of the center of $B_{\text{over-filled}}$ and “splitting” $B_{\text{over-filled}}$ 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 “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 $B_{\text{over-filled}}$ is improved by first merging $B_{\text{under-filled}}$ and then splitting $B_{\text{over-filled}}$. During the merge phase, the points in the $B_{\text{under-filled}}$ are released and are assigned to their next closest data bubble thereby emptying $B_{\text{under-filled}}$. $B_{\text{under-filled}}$ is re-positioned to the region of $B_{\text{over-filled}}$ by selecting a new seed $s_1$ for it from the current points in $B_{\text{over-filled}}$. Next, $B_{\text{over-filled}}$ 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 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.

5. 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 the figure) 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 closely 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 data bubbles identified in circles in part f in the figure). 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 from random to skewed ones. 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 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 in section 4.1 above) was set to 90% (we tested several experiments using a probability of 80% that did not change the quality of the resulting clustering structure). 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 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 [16]. The performance of OPTICS is determined using the F score measure [13] (where F = 2p*r/(p+r), p is precision and r is recall).

We notice from Table 1 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 with very small standard deviations for the ten experiments 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.

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 Table 1, 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.

Table 1. Performance evaluation of incremental data bubbles and the resulting clustering structure.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Scheme</th>
<th>Fscore</th>
<th>Compactness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Stand. deviation</td>
<td>Mean</td>
</tr>
<tr>
<td>Random2d</td>
<td>complete</td>
<td>0.7487</td>
<td>0.0479</td>
</tr>
<tr>
<td></td>
<td>Inc</td>
<td>0.8393</td>
<td>0.0395</td>
</tr>
<tr>
<td>Appear2d</td>
<td>complete</td>
<td>0.8398</td>
<td>0.0550</td>
</tr>
<tr>
<td></td>
<td>Inc</td>
<td>0.8586</td>
<td>0.0663</td>
</tr>
<tr>
<td>Disappear2d</td>
<td>complete</td>
<td>0.8186</td>
<td>0.0747</td>
</tr>
<tr>
<td></td>
<td>Inc</td>
<td>0.8377</td>
<td>0.0282</td>
</tr>
<tr>
<td>Extappear2d</td>
<td>complete</td>
<td>0.8076</td>
<td>0.0455</td>
</tr>
<tr>
<td></td>
<td>Inc</td>
<td>0.7946</td>
<td>0.0659</td>
</tr>
<tr>
<td>Gradmove2d</td>
<td>complete</td>
<td>0.7396</td>
<td>0.0475</td>
</tr>
<tr>
<td></td>
<td>Inc</td>
<td>0.6920</td>
<td>0.0870</td>
</tr>
<tr>
<td>Random10d</td>
<td>complete</td>
<td>0.8815</td>
<td>0.0026</td>
</tr>
<tr>
<td></td>
<td>Inc</td>
<td>0.8802</td>
<td>0.0000</td>
</tr>
<tr>
<td>Extappear10d</td>
<td>complete</td>
<td>0.9395</td>
<td>0.0253</td>
</tr>
<tr>
<td></td>
<td>Inc</td>
<td>0.9295</td>
<td>0.0376</td>
</tr>
<tr>
<td>Complex2d</td>
<td>complete</td>
<td>0.7329</td>
<td>0.0013</td>
</tr>
<tr>
<td></td>
<td>Inc</td>
<td>0.7114</td>
<td>0.0012</td>
</tr>
<tr>
<td>Complex5d</td>
<td>complete</td>
<td>0.6793</td>
<td>0.0018</td>
</tr>
<tr>
<td></td>
<td>Inc</td>
<td>0.6652</td>
<td>0.0022</td>
</tr>
<tr>
<td>Complex10d</td>
<td>complete</td>
<td>0.6496</td>
<td>0.0025</td>
</tr>
<tr>
<td></td>
<td>Inc</td>
<td>0.6465</td>
<td>0.0015</td>
</tr>
<tr>
<td>Complex20d</td>
<td>complete</td>
<td>0.6364</td>
<td>0.0042</td>
</tr>
<tr>
<td></td>
<td>Inc</td>
<td>0.6175</td>
<td>0.0038</td>
</tr>
</tbody>
</table>

Figure 9. Average percentage of rebuilt data bubbles vs percentage of points update.
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 10 shows the gain of using the triangle inequality in terms of the percentage of pruned distance computations when summarizing the complex 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 execute basically distance computations.

In addition, we notice in Figure 10 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 closely representative 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.

To compare the performance of our scheme of maintaining incremental data bubbles to a complete rebuilding of the data bubbles, we study the number of distance calculations that have to be computed in the incremental and complete rebuild methods. Figure 11 shows the average distance saving factor, which measures the fraction of the distance computations we save by using the incremental data bubbles with the triangle inequalities instead of the completely rebuilt ones without using the triangle inequalities on the complex database.

This factor also allows us to study the speed up we obtain with our incremental method for different amounts of updates in terms of percentage of the database size. We can see that we have significant speed up factors between 40 (for an update size of 10% of the database) up to approx. 200 for an update size of 2%. With a larger amount of updates, more new points are inserted into the database and are assigned to the incremental data bubbles (the complete rebuild method has to assign all the points in the database following every batch of insertions and deletions). Thus, as expected, we observe a decrease in the speed up factor as the set of updates becomes larger.

6. CONCLUSIONS

In this paper, 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.
There are several interesting directions for future research including compressing data streams of datasets with complex dynamics and distributions using incremental data bubbles, and investigating how to dynamically increase or decrease the number of incremental data bubbles to further improve the compression of a database.

7. ACKNOWLEDGEMENTS

- We would like to thank Nabil Salem for fruitful discussions in the early stages of the project.
- Research is partially funded by NSERC Canada.

8. REFERENCES


