Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering

Markus M. Breunig†, Hans-Peter Kriegel†, Peer Kröger†, Jörg Sander‡

† Institute for Computer Science
University of Munich
Oettingenstr. 67, D-80538 Munich, Germany
{ breunig | kriegel | kroegera }
@dbs.informatik.uni-muenchen.de

‡ Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4 Canada
jsander@cs.ubc.ca

ABSTRACT

In this paper, we investigate how to scale hierarchical clustering methods (such as OPTICS) to extremely large databases by utilizing data compression methods (such as BIRCH or random sampling). We propose a three step procedure: 1) compress the data into suitable representative objects; 2) apply the hierarchical clustering algorithm only to these objects; 3) recover the clustering structure for the whole data set, based on the result for the compressed data. The key issue in this approach is to design compressed data items such that not only a hierarchical clustering algorithm can be applied, but also that they contain enough information to infer the clustering structure of the original data set in the third step. This is crucial because the results of hierarchical clustering algorithms, when applied naively to a random sample or to the clustering features (CFs) generated by BIRCH, deteriorate rapidly for higher compression rates. This is due to three key problems, which we identify. To solve these problems, we propose an efficient post-processing step and the concept of a Data Bubble as a special kind of compressed data item. Applying OPTICS to these Data Bubbles allows us to recover a very accurate approximation of the clustering structure of a large data set even for very high compression rates. A comprehensive performance and quality evaluation shows that we only trade very little quality of the clustering result for a great increase in performance.

Keywords

Database Mining, Clustering, Sampling, Data Compression.

1. INTRODUCTION

Knowledge discovery in databases (KDD) is the non-trivial process of identifying valid, novel, potentially useful, and understandable patterns in large amounts of data. One of the primary data analysis tasks which should be applicable in this process is cluster analysis.

ACM SIGMOD 2001 May 21-24, Santa Barbara, California USA

There are different types of clustering algorithms for different types of applications. The most common distinction is between *partitioning* and *hierarchical* clustering algorithms (see e.g. [7]). Examples of partitioning algorithms are the *k*-means [8] and the *k*-medoids [7] algorithms which decompose a database into a set of *k* clusters. Most hierarchical clustering algorithms such as the single link method [9] and OPTICS [1] on the other hand compute a representation of the data set which reflects its hierarchical clustering structure. Whether or not the data set is then decomposed into clusters depends on the application.

In general, clustering algorithms do not scale well with the size of the data set. However, many real-world databases contain hundred thousands or even millions of objects. To be able to perform a cluster analysis of such databases, a very fast method is required (linear or near-linear runtime). Even if the database is medium sized, it makes a large difference for the user if he can cluster his data in a couple of seconds or in a couple of hours (e.g. if the analyst wants to try out different subsets of the attributes without incurring prohibitive waiting times). Therefore, improving clustering algorithms has received a lot of attention in the last few years.

A general strategy to scale-up clustering algorithms (without the need to invent a new cluster notion) is to draw a sample or to apply a kind of data compression (e.g. BIRCH [10]) before applying the clustering algorithm to the resulting representative objects. This approach is very effective for *k*-means type of clustering algorithms. For hierarchical clustering algorithms, however, the success of this approach is limited. Hierarchical clustering algorithms are based on the distances between data points which are not represented well by the distances between representative objects, especially when the compression rate increases.

In this paper, we analyze in detail the problems involved in the application of hierarchical clustering algorithms to compressed data. In order to solve these problems, we generalize the idea of a so-called Data Bubble introduced in [3] which is a more specialized kind of compressed data items, suitable for hierarchical clustering. We present two ways of generating Data Bubbles efficiently, either by using sampling plus a nearest neighbor classification or by utilizing BIRCH. Furthermore, we show that our method is efficient and effective in the sense that an extremely accurate approximation of the clustering structure for a very large data sets can be produced from a very small set of corresponding Data Bubbles. Thus, we achieve high quality clustering results for data sets containing hundred thousands of objects in a few minutes.

The rest of the paper is organized as follows. In section 2, we discuss data compression techniques for clustering, and give a short review of BIRCH. Hierarchical clustering is reviewed in section 3, including a short presentation of OPTICS. In section 4, we identify three key problems with a "naive" application of a hierarchical clustering algorithm to representative objects, called "size distortion", "lost objects", and "structural distortion". The size distortion problem and the lost objects problem have a rather straightforward solution which is presented in section 5. However, this solution can be fully effective only if the structural distortion problem is solved. For this purpose, the general concept of a Data Bubble is introduced in section 6. To recover the intrinsic clustering structure of an original data set even for extremely high compression rates, Data Bubbles integrate an estimation of the distance information needed by hierarchical clustering algorithms. In section 7, the notion of a Data Bubbles is specialized to Euclidean vector data in order to generate Data Bubbles very efficiently (by utilizing BIRCH or by drawing a sample plus a *k*-nearest neighbor classification). Section 8 presents an application of OPTICS to these Data Bubbles which indicates that all three problems are solved. In section 9, this observation is confirmed by a systematic experimental evaluation. Data sets of different sizes and dimensions are used to compare the clustering results for Data Bubbles with the results for the underlying data set. Section 10 concludes the paper.

2. DATA COMPRESSION FOR CLUSTERING

Random sampling is probably the most widely used method to "compress" a large data set in order to scale expensive data mining algorithms to large numbers of objects. The basic idea is rather simple: choose a subset of the database randomly and apply the data mining algorithm only to this subset instead of to the whole database. The hope is that if the number of objects sampled (the sample size) is large enough, the result of the data mining method on the sample will be similar enough to the result on the original database.

More specialized data compression methods have been developed recently to scale up *k*-means type clustering algorithms. The sufficient statistics intended to support clustering algorithms are basically the same for all these compression methods. As an example, we give a short description of BIRCH and discuss the major differences and the common features for the other methods in this section. BIRCH [10] uses a specialized tree-structure for clustering large sets of *d*-dimensional vectors. It incrementally computes compact descriptions of subclusters, called Clustering Features.

Definition 1: (Clustering Feature, *CF*)

Given a set of *n d*-dimensional data points $\{X_i\}$, $1 \le i \le n$.

The *Clustering Feature* (*CF*) for $\{X_i\}$ is defined as the triple $CF = (n, LS, ss)$, where $LS = \sum_{i=1\ldots n} X_i$ is the linear sum and $ss = \sum_{i=1\ldots n} X_i^2$ the square sum of the points.

The *CF*-values are sufficient to compute information about the sets of objects they represent like centroid, radius and diameter. They satisfy an important additivity condition, i.e. if $CF_1 = (n_1, LS_1, ss_1)$ and $CF_2 = (n_2, LS_2, ss_2)$ are the CFs for sets of points S_1 and S_2 respectively, then $CF_1 + CF_2 = (n_1 + n_2, LS_1 + LS_2, ss_1 + ss_2)$ is the clustering feature for the set $S_1 \cup S_2$.

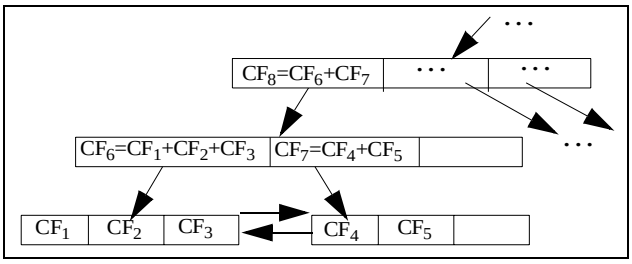

Figure 1: CF-tree structure

The CFs are organized in a balanced tree with branching factor *B* and a threshold *T* (see figure 1). A non-leaf node represents a subcluster consisting of all the subclusters represented by its entries. A leaf node has to contain at most *L* entries and the diameter of each entry in a leaf node has to be less than *T*.

BIRCH performs a sequential scan over all data points and builds a *CF*-tree similar to the construction of B^+-trees. A point is inserted by inserting the corresponding *CF*-value into the closest leaf. If an entry in the leaf can absorb the new point without violating the threshold condition, its *CF* is updated. Otherwise, a new entry is created in the leaf node, and, if the leaf node then contains more than *L* entries, it and maybe its ancestors are split. A clustering algorithm can then be applied to the entries in the leaf nodes of the *CF*-tree. The number of leaf nodes contained in a *CF*-tree can be specified by a parameter in the original implementation.

In [2] another compression technique for scaling up clustering algorithms is proposed. Their method produces basically the same type of compressed data items as BIRCH, i.e. triples of the form (*n*, *LS*, *ss*) as defined above. The method is, however, more specialized to *k*-means type clustering algorithms than BIRCH in the sense that the authors distinguish different sets of data items: A set of compressed data items *DS* which is intended to condense groups of points unlikely to change cluster membership in the iterations of the (*k*-means type) clustering algorithm, a set of compressed data items *CS* which represents tight subclusters of data points, and a set of regular data points *RS* which contains all points which cannot be assigned to any of the compressed data items. While BIRCH uses the diameter to threshold compressed data items, [2] apply different threshold conditions for the construction of compressed data items in the sets *DS* and *CS* respectively.

A very general framework for compressing data has been introduced recently in [4]. Their technique is intended to scale up a large collection of data mining methods. In a first step, the data is grouped into regions by partitioning the dimensions of the data. Then, in the second step, a number of moments are calculated for each region induced by this partitioning (e.g. means, minima, maxima, second order moments such as X_i^2 or X_iX_j, and higher order moments depending on the desired degree of approximation). In the third step, they create for each region a set of squashed data items so that its moments approximate those of the original data falling in the region. Obviously, information such as clustering features for the constructed regions, to speed-up *k*-means type clustering algorithms, can be easily derived from this kind of squashed data items.

For the purpose of clustering, we can also compute sufficient statistics of the form (*n*, *LS*, *ss*) efficiently based on a random sample since we can assume that a distance function is defined for the objects in the data set. This allows us to partition the data set using a

k-nearest neighbor classification. This method has the advantages that we can control exactly the number of representative objects for a data set and that we do not rely on other parameters (like diameter, or bin-size) to restrict the size of the partitions for representatives given in the form (*n*, *LS*, *ss*). The method works as follows:

1. Draw a random sample of size *k* from the database to initialize *k* sufficient statistics.
2. In one pass over the original database, classify each original object *o* to the sampled object *s* it is closest to and incrementally add *o* to the sufficient statistics initialized by *s*, using the additivity condition given above.

The application of *k*-means type clustering algorithms to compressed data items (*n*, *LS*, *ss*) is rather straightforward. The *k*-means algorithm represents clusters by the mean of the points contained in that cluster. It starts with an assignment of data points to *k* initial cluster centers, resulting in *k* clusters. Then it iteratively performs the following steps while the cluster centers change: 1) Compute the mean for each cluster. 2) Re-assign each data point to the closest of the new cluster centers. When using sufficient statistics the algorithm just has to be extended so that it treats the triplets (*n*, *LS*, *ss*) as data points *LS*/*n* with a weight of *n* when computing cluster means, i.e. the mean of *m* compressed points LS_1/n_1, ..., LS_m/n_m is calculated as $(LS_1/n_1 + ... + LS_m/n_m) / n_1 + ... + n_m$.

3. HIERARCHICAL CLUSTERING

Typically, hierarchical clustering algorithms represent the clustering structure of a data set *D* by a *dendrogram*, i.e. a tree that iteratively splits *D* into smaller subsets until each subset consists of one object. In such a hierarchy, each node of the tree represents a cluster of *D*. The dendrogram can either be created bottom-up (*agglomerative approach*) or top-down (*divisive approach*) by merging, respectively dividing clusters at each step.

There are a lot of different algorithms producing the same hierarchical structure (see e.g. [9], [6]). In general, they are based on the inter-object distances and on finding the nearest neighbors of objects and clusters. Therefore, the runtime complexity of these clustering algorithms is at least $O(n^2)$, if all inter-object distances for an object have to be checked to find its nearest neighbor. Agglomerative hierarchical clustering algorithms, for instance, basically keep merging the closest pairs of objects to form clusters. They start with the "disjoint clustering" obtained by placing every object in a unique cluster. In every step the two "closest" clusters in the current clustering are merged. For this purpose, they define a distance measure for sets of objects. For the so-called "single link method", for example the distance between two sets of objects is defined as the minimal distance between their objects (see figure 2 for an illustration of the single link method).

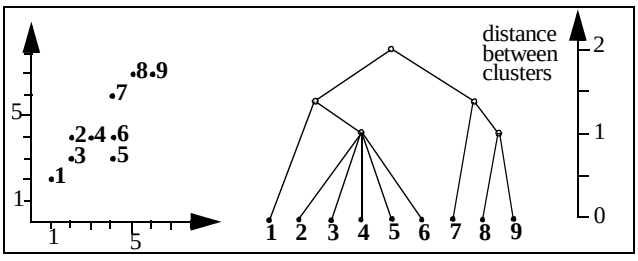

Figure 2: Single link clustering of n = 9 objects

OPTICS [1] is another hierarchical clustering method that has been proposed recently. This method is based on a different algorithmic approach which reduces some of the shortcomings of traditional hierarchical clustering algorithms. It weakens the so called "single link effect"; it computes information, that can be displayed in a diagram that is a more appropriate for very large data sets than a dendrogram; and it is specifically designed to be based on range queries, which can be efficiently supported by index-based access structures. This results in a runtime complexity of O(*n* log *n*) under the condition that the underlying index structure works well.

In the following we give a short review of OPTICS [1], since we will use this algorithm to evaluate our method for hierarchical clustering using compressed data items. The method itself can be easily adapted to work with classical hierarchical clustering algorithms as well.

First, the basic concepts of neighborhood and nearest neighbors are defined in the following way.

Definition 2: (ε-neighborhood and *k*-distance of an object *P*)

Let *P* be an object from a database *D*, let ε be a distance value, let *k* be a natural number and let *d* be a distance metric on *D*. Then, the ε-neighborhood $N_\varepsilon(P)$ is a set of objects *X* in *D* with $d(P,X) \le \varepsilon$:

$$N_\varepsilon(P) = \{ X \in D \mid d(P,X) \le \varepsilon \},$$

and the *k*-distance of *P*, *k-dist*(*P*), is the distance $d(P, O)$ between *P* and an object $O \in D$ such that at least for *k* objects $O' \in D$ it holds that $d(P, O') \le d(P, O)$, and for at most *k*-1 objects $O' \in D$ it holds that $d(P, O') < d(P, O)$. Note that *k*-dist(*P*) is unique, although the object *O* which is called 'the' *k*-nearest neighbor of *P* may not be unique. When clear from the context, $N_k(P)$ is used as a shorthand for $N_{k\text{-}dist(P)}(P)$.

The objects in the set $N_k(P)$ are called the "*k*-nearest-neighbors of *P*" (although there may be more than *k* objects contained in the set if the *k*-nearest neighbor of *P* is not unique).

OPTICS is based on a density-based notion of clusters introduced in [5]. For each object of a density-based cluster, the ε-neighborhood has to contain at least a minimum number of objects. Such an object is called a *core object*. Clusters are defined as maximal sets of density-connected objects. An object *P* is density-connected to *Q* if there exists an object *O* such that both *P* and *Q* are density-reachable from *O* (directly or transitively). *P* is directly density-reachable from *O* if $P \in N_\varepsilon(O)$ and *O* is a core object. Thus, a "flat" partitioning of a data set into a set of clusters is defined, using *global* density parameters. OPTICS extends this density-based clustering approach to create an augmented *ordering* of the database representing its density-based clustering structure. The cluster-ordering contains information which is equivalent to the density-based clusterings corresponding to a broad range of parameter settings. This cluster-ordering of a data set is based on the notions of "core-distance" and "(density-)reachability-distance".

Definition 3: (core-distance of an object *P*)

Let *P* be an object from a database *D*, let ε be a distance value and let *MinPts* be a natural number. Then, the *core-distance* of *P* is defined as

$$core\text{-}dist_{\varepsilon,MinPts}(P) = \begin{cases} \infty, \text{ if } |N_\varepsilon(P)| < MinPts \\ MinPts\text{-}dist(P), \text{ otherwise} \end{cases}.$$

The core-distance of an object P is the smallest distance $\varepsilon' \le \varepsilon$ such that P is a core object with respect to ε' and *MinPts* - if such a distance exists, i.e. if there are at least *MinPts* objects within the ε-neighborhood of P. Otherwise, the core-distance is ∞.

Definition 4: (reachability-distance of an object P w.r.t. O)

Let P and O be objects, $P \in N_\varepsilon(O)$, let ε be a distance value and *MinPts* be a natural number. Then, the *reachability-distance* of P with respect to O is defined as

$reach\text{-}dist_{\varepsilon,MinPts}(P,O) = max(core\text{-}dist_{\varepsilon,MinPts}(O), d(O,P))$.

Intuitively, *reach-dist*(P,O) is the smallest distance such that P is directly density-reachable from O if O is a core object. Therefore *reach-dist*(P,O) cannot be smaller than *core-dist*(O) because for smaller distances no object is directly density-reachable from O. Otherwise, if O is not a core object, *reach-dist*(P,O) is ∞. (See figure 3.)

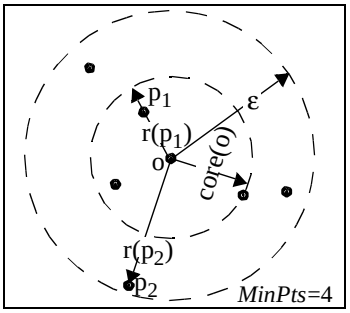

Figure 3: core-dist(O), reach-dists r(P_1,O), r(P_2,O)

Using the core- and reachability-distances, OPTICS computes a "walk" through the data set, and assigns to each object O its core-distance and the smallest reachability-distance *reachDist* with respect to an object considered *before* O in the walk (see [1] for details). This walk satisfies the following condition: Whenever a set of objects C is a density-based cluster with respect to *MinPts* and a value ε' smaller than the value ε used in the OPTICS algorithm, then the objects of C (possibly without a few border objects) form a subsequence in the walk. The *reachability-plot* consists of the reachability values (on the *y*-axis) of all objects, plotted in the ordering which OPTICS produces (on the x-axis). This yields an easy to understand visualization of the clustering structure of the data set. The "dents" in the plot represent the clusters because objects within a cluster typically have a lower reachability-distance than objects outside a cluster. A high reachability-distance indicates a noise object or a jump from one cluster to another cluster.

Figure 4 shows the reachability-plot for two 2-dimensional synthetic data sets, DS1 and DS2, which we will use in the following sections to evaluate our approach. DS1 contains one million points grouped into several nested clusters of different densities and distributions (uniform and Gaussian) and noise objects. DS2 contains 100,000 objects in 5 Gaussian clusters of 20,000 objects each. The figure also shows the result of applying the basic OPTICS algorithm to these data sets. The "dents" in the plots represent the clusters, clearly showing the hierarchical structure for DS1.

4. PROBLEMS WITH A NAIVE APPLICATION TO RANDOM SAMPLES OR TO CF CENTERS

When we want to apply a hierarchical clustering algorithm to a "compressed" data set, it is not clear whether we will get satisfactory results if we treat clustering features (n, LS, ss) as data points LS/n, or if we simply use a random sample of the database. Hierarchical clustering algorithms do not compute any cluster centers but

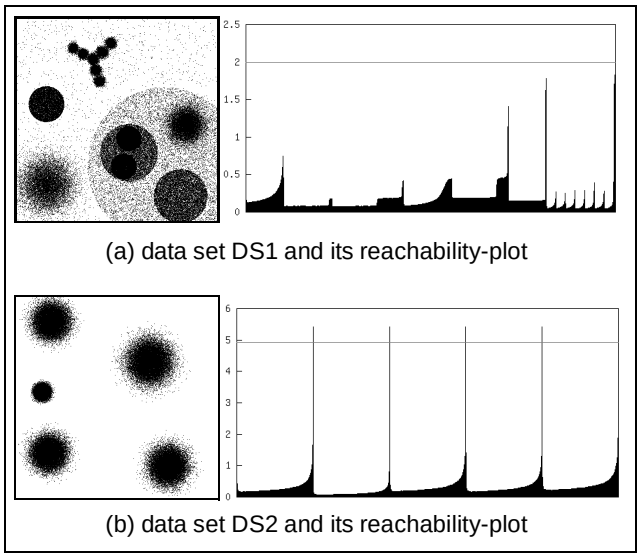

Figure 4: Databases DS1 and DS2 and the original OPTICS reachability-plots. The runtimes using the basic OPTICS-algorithm were 16,637sec and 993sec.

compute a special representation of the distances between points and between clusters. This information, however, may not be well reflected by a reduced set of points such as cluster feature centers or random sample points. We present this application to discuss the major problems involved in hierarchical clustering of compressed data sets. The algorithmic schema for the application of OPTICS to both CFs and a random sample is depicted in figure 5.

We assume that the number of representative objects k is small enough to fit into main memory. We will refer to these algorithms as "OPTICS-CF_{naive}" and "OPTICS-SA_{naive}" for the naive application of OPTICS to *CF*s and to a random *SA*mple, respectively. Figure 6 shows the results of the algorithms OPTICS-SA_{naive} and OPTICS-CF_{naive} on DS1 for three different sample sizes: 10,000 objects, 1,000 objects and 200 objects. For the large number of representative objects (10,000 objects, i.e. compression factor 100), the quality of the reachability-plot of OPTICS-SA_{naive} is comparable to the quality of applying OPTICS to the original database. For small values of k, however, the quality of the result suffers considerably. For a compression factor of 1,000, the hierarchical clustering structure of the database is already distorted, for a compression factor of 5,000, the clustering structure is almost completely lost. The results are even worse for OPTICS-CF_{naive}: none of the reachability-plots even crudely represents the clustering structure of the database. We will call this problem "*structural distortions*".

Figure 7 shows the results on DS2 for both algorithms for 100 representative objects. For larger number of representative objects the

1. Either (CF): Execute BIRCH and extract the centers of the k leaf CFs as representative objects.
   Or (SA): Take a random sample of k objects from the database as representative objects.
2. Optional: Build an index for the representative objects (used by OPTICS to speed up range queries).
3. Apply OPTICS to the representative objects.

Figure 5: Algorithm OPTICS-CF_{naive} and OPTICS-SA_{naive}

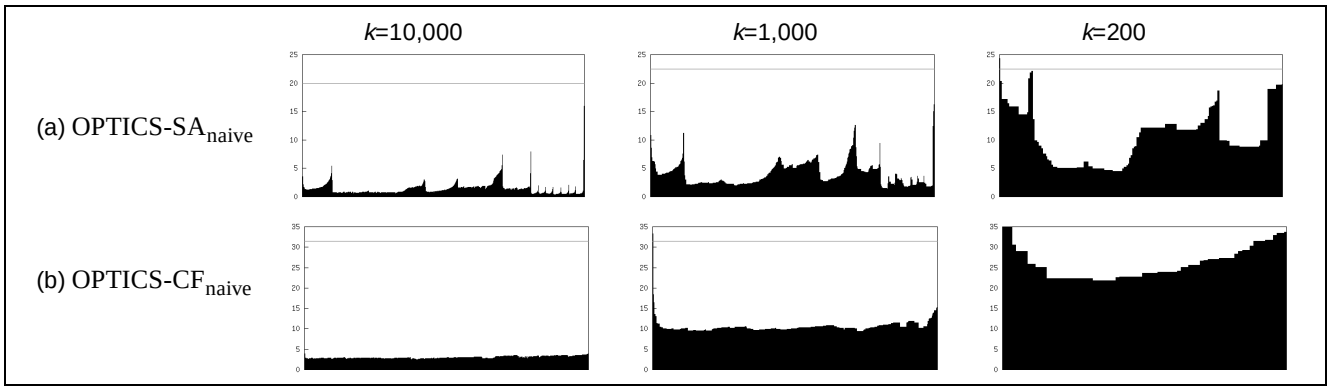

Figure 6: DS1-results of OPTICS-SA$_{naive}$ an OPTICS-CF$_{naive}$ for 10,000, 1,000 and 200 representative objects

results of both algorithms are quite good, due to fact that the clusters in this data set are well separated. However, even for such simple data sets as DS2 the results of a naive application deteriorate with high compression rates. OPTICS-SA$_{naive}$ preserves at least the information that 5 clusters exist while OPTICS-CF$_{naive}$ loses one cluster. But for both algorithms, we see that the sizes of the clusters are distorted, i.e. some clusters seem larger than they really are and others seem smaller. The reachability-plots are stretched and squeezed. We will call this problem "*size distortions*".

Apart from the problems discussed so far, there is another problem if we want to apply clustering as one step in a larger knowledge discovery effort, in which the objects are first assigned to clusters and then further analyzed: we do not have direct clustering information about the database objects. Only *some* (in case of sampling) or even *none* (when using CFs) of the database objects are contained in the reachability-plot. This problem will be called "*lost objects*".

In order to apply hierarchical clustering algorithms to highly compressed Data, we have to solve these three problems. We will see that the size distortion problem and the lost objects problem have a rather straightforward solution. However, solving these problems in isolation improves the clustering results only by a minor degree. The basic problem is the structural distortion which requires a more sophisticated solution.

5. SOLVING THE SIZE DISTORTION AND THE LOST OBJECT PROBLEM

In order to alleviate the problem of size distortions, we can weigh each representative object with the number *n* of objects they actually represent. When plotting the final cluster ordering we can simply repeat the reachability value for a representative object *n* times, which corrects the observed stretching and squeezing in the reachability-plots. (Note that we can apply an analogous technique to expand a dendrogram produced by other hierarchical algorithms.)

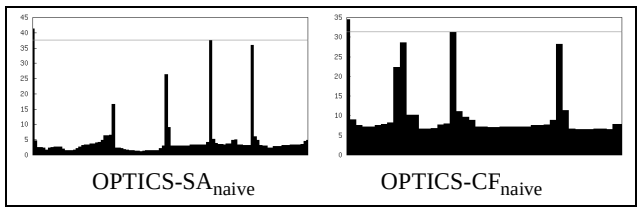

Figure 7: DS2-results of OPTICS-SA$_{naive}$ and OPTICS-CF$_{naive}$ for 100 objects

When using BIRCH, the weight *n* of a representative object is already contained in a clustering feature (*n, LS, ss*). When using a random sample, we can easily determine theses numbers for the sample points by classifying each original object to the sample point which is closest to it (using a nearest-neighbor classification).

This solution to the size distortion problem also indicates how to solve the lost objects problem. The idea is simply to apply the classification step not only in the sampling based approach but also to clustering features to determine the objects which actually "belong" to a representative object. When generating the final cluster ordering, we store with the reachability values that replace the values for each representative object s_j the identifiers of the original objects classified to s_j. By doing so, we, on the one hand, correct the stretching and squeezing in the reachability-plot, i.e. we solve the size distortions problem, and, on the other hand, insert all original objects into the cluster ordering, thus solving the lost objects problem at the same time. The algorithmic schema for both methods is given in figure 8.

We will refer to these algorithms as "OPTICS-CF$_{weighted}$" and "OPTICS-SA$_{weighted}$", depending on whether we use OPTICS with weighted CFs or with weighted random sample points. The difference to the naive schema lies only in step 4 and 5 where we do the *nn*-classification and adapt the reachability plot. We read each original object o_i and classify it by executing a nearest neighbor query in the sampled database. If we have built an index on the sampled database in step 2, we can reuse it here. To understand step 5, let the nearest neighbor of o_i be s_j. We set the core-distance of o_i to the

1. Either (CF): Execute BIRCH and extract the centers of the *k* leaf CFs as representative objects.
   Or (SA): Take a random sample of *k* objects from the database as representative objects.
2. Optional: Build an index for the representative objects (used by OPTICS to speed-up range queries).
3. Apply OPTICS to the representative objects.
4. For each database object compute the representative object it is closest to (using a nearest-neighbor query).
5. Replace the representative objects by the corresponding sets of original objects in the reachability plot.

Figure 8: Algorithm OPTICS-CF$_{weighted}$ and algorithm OPTICS-SA$_{weighted}$

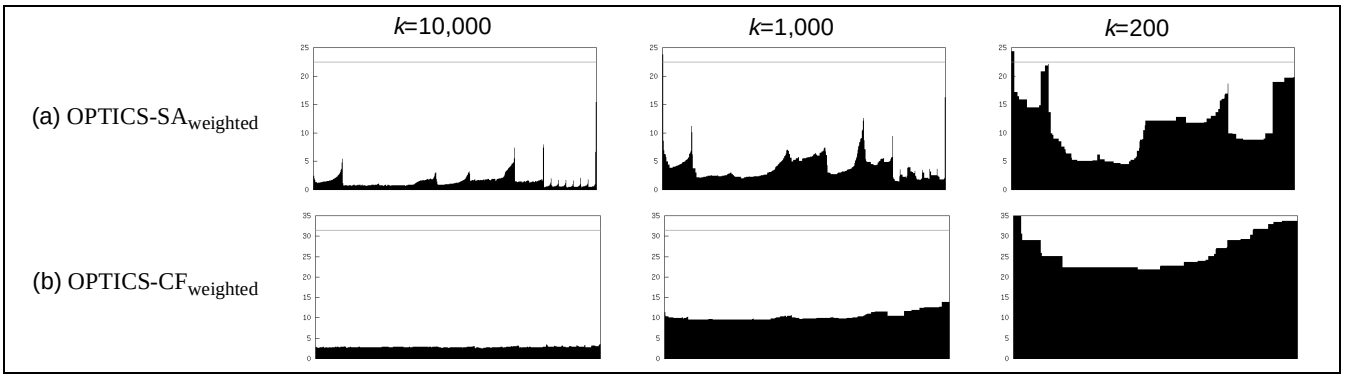

Figure 9: DS1-results of OPTICS-SA$_{weighted}$ an OPTICS-CF$_{weighted}$ for 10,000, 1,000 and 200 representative objects

core-distance of s_j and the position of o_i in the reachability plot to the position of s_j plus the number of objects which have already been classified to s_j. If o_i is the first object classified to s_j, we set the reachability of o_i to the reachability of s_j, otherwise we set o_i.*reachDist* to min{s_j.*reachDist*, (s_j+1).*reachDist*}. The motivation for this is that s_j.*reachDist* is the reachability we need to first get to s_j but once we are there, the reachabilities of the other objects will be approximately the same as the reachability of the next object in the cluster ordering of the sample. Then we write o_i back to disc. Thus, we make one pass (reading and writing) over the original database. Finally, we sort the original database according to the position numbers, thus bringing the whole database into the cluster ordering.

Figure 9(a) shows the results for DS1 of the OPTICS-SA$_{weighted}$ for three different sample sizes: 10,000 objects, 1,000 objects and 200 objects. Figure 9(b) shows the same information for the OPTICS-CF$_{weighted}$ algorithm. The results of both algorithms look very similar to the results of the naive versions of the algorithms. Although we have corrected the size distortion in both cases, the structural distortion dominates the visual impression. Both versions, however, have the advantage that all original database objects are now actually represented in the cluster ordering.

That the post-processing step really solves the size distortions problem is visible in figure 10 which shows the results for DS2. The result of OPTICS-SA$_{weighted}$ is quite good: all five clusters are clearly visible and have the correct sizes. The cluster ordering generated by OPTICS-CF$_{weighted}$ has also improved as compared to OPTICS-SA$_{naive}$. Obviously, post-processing alleviates the size distortion problem and solves the lost objects problem. However, the lost cluster cannot be recovered by OPTICS-CF$_{weighted}$. Weighing the representative objects and classifying the database can be fully effective only when we solve the structural distortion problem.

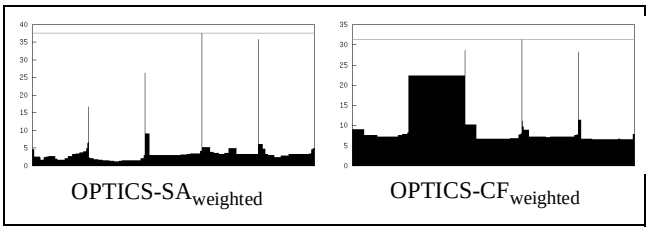

Figure 10: DS2-results of OPTICS-SA$_{weighted}$ and OPTICS-CF$_{weighted}$ for 100 objects

6. DATA BUBBLES AND ESTIMATED DISTANCES: SOLVING THE STRUCTURAL DISTORTION PROBLEM

The basic reason for the structural distortion problem when using very high compression rates is that the distance between the original data points is not represented well by only the distance between representative objects. Figure 11 illustrates the problem using two extreme situations. In these cases, the distance between the representative points *rA* and *rB* is the same as the distance between the representative points *rC* and *rD*. However, the distance between the corresponding sets of points which they actually represent is very different. This error is one source for the structural distortion. A second source for structural distortion is the fact that the true distances (and hence the true *reachDists*) for the points within the point sets are very different from the distances (and hence the *reachDists*) we compute for their representatives. This is the reason why it is not possible to recover clusters by simply weighing the representatives with the number of points they represent. Weighing only stretches certain parts of the reachability-plot by using the *reachDist* values of the representatives. For instance, assume that the *reachDist* values for the representative points are as depicted in figure 11. When expanding the plot for *rD*, we assign to the first objects classified to belong to *rD* the *reachDist* value *reachDist*(*rD*). Every other object in *D* is then assigned the value *reachDist*(*rY*) which is, however, almost the same as the value *reachDist*(*rD*). Weighing the representative object will be more effective, if we use at least a close estimate of the true reachability values for the ob-

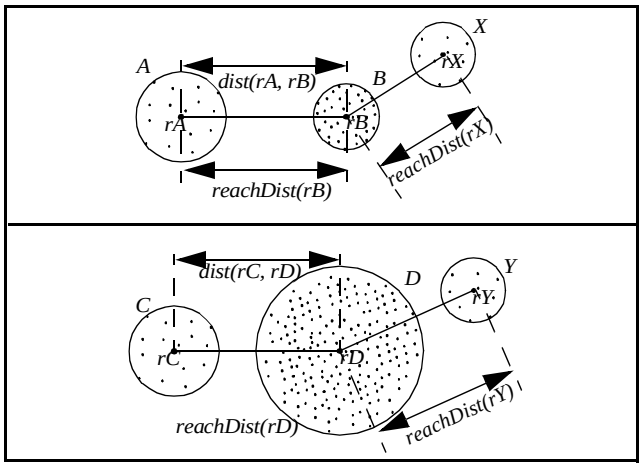

Figure 11: Illustration for the structural distortion problem

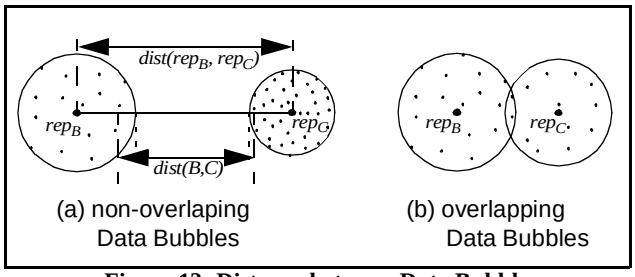

Figure 12: Distance between Data Bubbles

jects in a data set. Only then, we will be able to recover the whole cluster *D*: the reachability value for *rD* would then be expanded by a sequence of very small reachability values which appear as a "dent" (indicating a cluster) in the reachability plot.

To solve the structural distortion problem we need a better distance measure for compressed data items and we need a good estimation of the true reachability values within sets of points. To achieve this goal, we first introduce the concept of Data Bubbles summarizing the information about point sets which is actually needed by a hierarchical clustering algorithm to operate on. Then we give special instances of such Data Bubbles for Euclidean vector spaces and show how to construct a cluster ordering for a very large data set using only a very small number of Data Bubbles. We define Data Bubbles as a convenient abstraction summarizing the sufficient information on which hierarchical clustering can be performed.

Definition 5: (Data Bubble)

Let $X=\{X_i\}$ $1 \le i \le n$ be a set of n objects.

Then, the *Data Bubble B* w.r.t. X is defined as a tuple $B_X = (rep, n, extent, nnDist)$, where

- *rep* is a representative object for X (which may or may not be an element of X);
- n is the number of objects in X;
- *extent* is a real number such that "most" objects of X are located within a "radius" *extend* around *rep*;
- *nnDist* is a function denoting the estimated average k-nearest neighbor distances within the set of objects X for some values k, k=1, ..., k = *MinPts*. A particular expected knn-distance in B_X is denoted by $nnDist(k, B_X)$.

Using the radius *extent* and the expected nearest neighbor distance, we can define a distance measure between two Data Bubbles that is suitable for hierarchical clustering.

Definition 6: (distance between two Data Bubbles)

Let $B=(rep_B, n_B, e_B, nnDist_B)$ and $C=(rep_C, n_C, e_C, nnDist_C)$ be two Data Bubbles.

Then, the *distance* between B and C is defined as $dist(B, C)$ =

$$\begin{cases} 0 & \text{if } B = C \\ dist(rep_B, rep_C) - (e_B + e_C) + nnDist(1, B) + nnDist(1, C) & \\ \qquad \text{if } dist(rep_B, rep_C) - (e_1 + e_2) \ge 0 & \\ max(nnDist(1, B), nnDist(1, C)) & \text{otherwise} \end{cases}$$

Besides the case that $B = C$ (in which the distance obviously has to be 0), we have to distinguish the two cases shown in figure 12. The distance between two non-overlapping Data Bubbles is the distance of their centers minus their radii plus their expected nearest neighbor distances. If the Data Bubbles overlap, we take the maximum of their expected nearest neighbor distances as their distance. Intuitively, this distance definition is intended to approximate the distance of the two closest points in the Data Bubbles.

When applying a classical hierarchical clustering algorithm such as the single link method to Data Bubbles, we do not need more information than defined above. For the algorithm OPTICS, however, we have to define additionally the appropriate notion of a core-distance and a reachability-distance.

Definition 7: (core-distance of a Data Bubble *B*)

Let $B=(rep_B, n_B, e_B, nnDist_B)$ be a Data Bubble, let ε be a distance value, let *MinPts* be a natural number and let $N = \{X \mid dist(B, X) \le \varepsilon\}$. Then, the *core-distance* of B is defined as

$$core\text{-}dist_{\varepsilon, MinPts}(B) = \begin{cases} \infty & \text{if } \left(\sum_{X=(r,n,e,d) \in N} n \right) < MinPts \\ dist(B, C) + nnDist(k, C) & \text{otherwise} \end{cases},$$

where C and k are given as follows: $C \in N$ has maximal $dist(B, C)$ such that $\sum_{X \in N,\ dist(B,X) < dist(B,C)} n < MinPts$, and $k = MinPts - \sum_{X \in N,\ dist(B,X) < dist(B,C)} n$.

This definition is based on a similar notion as the core-distance for data points. For points, the core-distance is ∞ if the number of points in the ε-neighborhood is smaller than *MinPts*. Analogously, the core-distance for Data Bubbles is ∞ if the sum of the numbers of points represented by the Data Bubbles in the ε-neighborhood is smaller than *MinPts*. For points, the core-distance (if not ∞) is the distance to the *MinPts*-neighbor. For Data Bubble B, it is the estimated *MinPts*-distance for the representative rep_B of B. Data Bubbles usually summarize at least *MinPts* points. Note that in this case, the $core\text{-}dist_{\varepsilon, MinPts}(B)$ is equal to $nnDist(MinPts, B)$ according to the above definition. Only in very rare cases, or when the compression rate is extremely low, a Data Bubble may represent less than *MinPts* points. In this case we estimate the *MinPts*-distance of rep_B by taking the distance between B and the closest Data Bubble C so that B and C and all Data Bubbles which are closer to B than C contain together at least *MinPts* points. To this distance we then add an estimated nearest k-neighbor distance in C, where k is computed by subtracting from *MinPts* the number of points of all Data Bubbles which are closer to B than C (which by selection of C do not add up to *MinPts*).

Given the core-distance, the reachability-distance for Data Bubbles is defined in the same way as the reachability-distances on data points.

Definition 8: (reachability-distance of a Data Bubble *B* w.r.t. Data Bubble *C*)

Let $B=(rep_B, n_B, e_B, nnDist_B)$ and $C=(rep_C, n_C, e_C, nnDist_C)$ be Data Bubbles, let ε be a distance value, let *MinPts* be a natural number, and let $B \in N_C$, where $N_C = \{X \mid dist(C, X) \le \varepsilon\}$.

Then, the *reachability-distance* of B w.r.t. C is defined as $reach\text{-}dist_{\varepsilon, MinPts}(B,C) = max(core\text{-}dist_{\varepsilon, MinPts}(C), dist(C,B))$.

Using these distances, OPTICS can be applied to Data Bubbles in a straight forward way. However, we also have to change the values which replace the *reachDist* values of the representative objects when generating the final reachability plot. When replacing the *reachDist* for a Data Bubble B, we plot for the first original object

the *reachDist* of *B* (marking the jump to *B*) followed by (*n*-1)-times an estimated reachability value for the *n*-1 remaining objects that *B* describes. This estimated reachability value is called the "virtual reachability of *B*", and is defined as follows:

Definition 9: (virtual reachability of a Data Bubble *B*)
Let $B = (rep_B, n_B, e_B, nnDist_B)$ be a Data Bubble and *MinPts* a natural number. The *virtual reachability* of the n_B points described by *B* is then defined as

$$virtual\text{-}reachability(B) = \begin{cases} nnDist_B(MinPts, B) & \text{if } n_B \geq MinPts \\ \text{core-}dist(B) & \text{otherwise} \end{cases}.$$

The intuitive idea is the following: if we assume that the points described by *B* are more or less uniformly distributed in a sphere of radius e_B around rep_B, and *B* describes at least *MinPts* points, the true *reachDist* of most of these points would be close to their *MinPts*-nearest neighbor distance. If, on the other hand, *B* contains less than *MinPts* points, the true *reachDist* of any of these points would be close to the core-distance of *B*.

7. DATA BUBBLES FOR EUCLIDEAN VECTOR SPACES

Data Bubbles provide a very general framework for applying a hierarchical clustering algorithm, and in particular OPTICS, to compressed data items created from an arbitrary data set - assuming only that a distance function is defined for the original objects. In the following, we will specialize these notions and show how Data Bubbles can be *efficiently* created for data from Euclidean vector spaces using sufficient statistics (*n*, *LS*, *ss*).

To create a Data Bubble B_X=(*rep*, *n*, *extent*, *nnDist*) for a set *X* of *n* *d*-dimensional data points, we have to determine the components in B_X. A natural choice for the representative object *rep* is the mean of the vectors in *X*. If these points are approximately uniformly distributed around the mean *rep*, the average pairwise distance between the points in *X* is a good approximation for a radius around *rep* which contains most of the points in *X*. Under the same assumption, we can also compute the expected *k*-nearest neighbor distance of the points in *B* in the following way:

Lemma 1: (expected *k-nn* distances for Euclidean vector data)
Let *X* be a set of *n* *d*-dimensional points. If the *n* points are uniformly distributed inside a sphere with center *c* and radius *r*, then the *expected k-nearest neighbor distance inside X* is equal to

$$\left(\frac{k}{n}\right)^{1/d} \cdot r .$$

Proof: The volume of a *d*-dimensional sphere of radius *r* is $V_S(r) = \frac{\sqrt{\pi^d}}{\Gamma\left(\frac{d}{2}+1\right)} \cdot r^d$ (Γ is the Gamma-Function). If the *n* points are uniformly distributed inside the sphere, we expect one point in the volume $V_S(r) / n$ and *k* points in the volume $k\ V_S(r) / n$. Thus, the expected *k*-nearest neighbor distance is equal to a radius r' of a sphere having this volume $k\ V_S(r) / n$. By simple algebraic transformations it follows that $r' = \left(\frac{k}{n}\right)^{1/d} \cdot r$. ♦

Using these notions, we can define a Data Bubble for a set of Euclidean vector data in the following way:

1. Either (CF): execute BIRCH and extract the CFs.
   Or (SA): sample *k* objects from the database randomly and initialize *k* sufficient statistics.
   Classify the original objects to the closest sample object, computing sufficient statistics.
   Save classification information for use in the last step.
2. Compute Data Bubbles from the sufficient statistics.
3. Apply OPTICS to the Data Bubbles.
4. If (CF): classify the original objects to the closest Data Bubble.
5. Replace the Data Bubbles by the corresponding sets of original objects.

Figure 13: Algorithm OPTICS-$CF_{Bubbles}$ and algorithm OPTICS-$SA_{Bubbles}$

Definition 10: (Data Bubble for Euclidean vector data)
Let $X=\{X_i\}$, $1 \leq i \leq n$ be a set of *n* *d*-dimensional data points. Then, a *Data Bubble* B_X for *X* is given by the tuple B_X = (*rep*, *n*, *extent*, *nnDist*), where

$rep = \left(\sum_{i=1\ldots n} X_i\right) / n$ is the *center* of *X*,

$extent = \sqrt{\frac{\sum_{i=1..n}\sum_{j=1..n}(X_i - X_j)^2}{n \cdot (n-1)}}$ is the *radius* of *X*, and

nnDist is defined by $nnDist(k, B) = \left(\frac{k}{n}\right)^{1/d} \cdot extent$.

Data Bubbles can be generated in many different ways. Given a set of objects *X*, they can be straight forwardly computed. Another possibility is to compute them from sufficient statistics (*n*, *LS*, *ss*) as defined in definition 1:

Corollary 1:
Let B_X = (*rep*, *n*, *extent*, *nnDist*) be a Data Bubble for a set $X=\{X_i\}$, $1 \leq i \leq n$, of *n* *d*-dimensional data points. Let *LS* be the linear sum and *ss* the square sum of the points in *X*.

Then, $rep = \frac{LS}{n}$, and $extent = \sqrt{\frac{2 \cdot n \cdot ss - 2 \cdot LS^2}{n \cdot (n-1)}}$.

For our experimental evaluation of Data Bubbles we compute them from sufficient statistics (*n*, *LS*, *ss*). One algorithm is based on the CFs generated by BIRCH, the other algorithm is based on random sampling, as described in section 2.

8. CLUSTERING HIGHLY COMPRESSED DATA USING DATA BUBBLES

In this section, we present the algorithms OPTICS-$CF_{Bubbles}$ and OPTICS-$SA_{Bubbles}$ to evaluate whether or not our Data Bubbles actually solve the structural distortion problem. OPTICS-$CF_{Bubbles}$ uses Data Bubbles which are computed from the leaf CFs of a CF-tree created by BIRCH. OPTICS-$SA_{Bubbles}$ uses Data Bubbles which are computed from sufficient statistics based on a random sample plus nn-classification. Both algorithms are presented using again one algorithmic schema, which is given in figure 13.

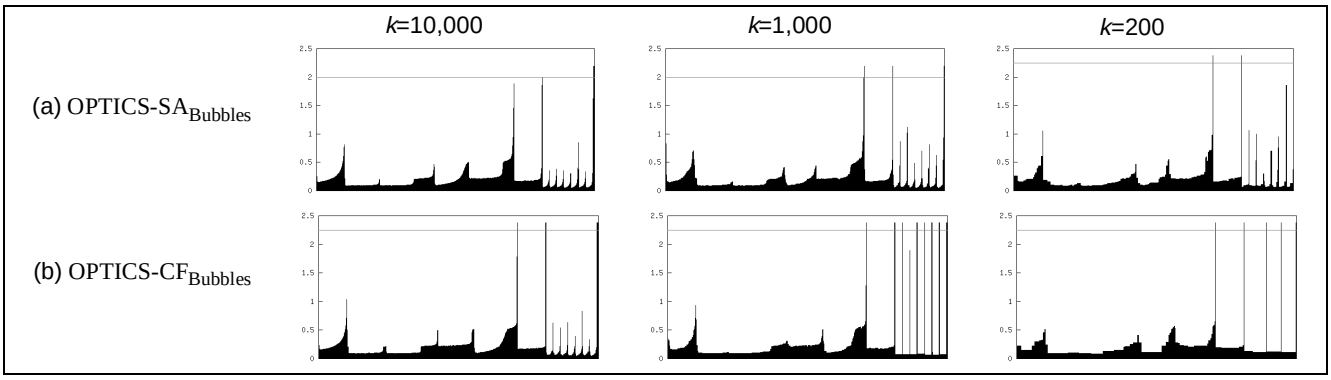

Figure 14: DS1-results for 10,000, 1,000 and 200 representative objects

Step 1 is different for the two algorithms. For OPTICS-$CF_{Bubbles}$ we execute BIRCH, and extract the CFs from the leaf nodes of the CF-tree. For OPTICS-$SA_{Bubbles}$ we draw a random sample of size k and initialize a tuple (n, LS, ss) for each sampled object s with this object, i.e. n=1, LS=s and ss equals the square sum of s. Then, we read each object o_i from the original database, classify o_i to the sample object it is closest to, and update (n, LS, ss) for the corresponding sample point. We save the classification information by writing it to a file, as we can use it again in step 5. This is cheaper than to re-do the classification.

In step 2 and 3, we compute Data Bubbles from the sufficient statistics by applying Corollary 1, and apply OPTICS to them. Because of the rather complex distance measure between Data Bubbles, we cannot use an index to improve the time complexity of this step and it runs in O(k*k). However, the purpose of our approach is to make k very small so that this is acceptable.

Step 4 applies to OPTICS-$CF_{Bubbles}$ only, as we do not have information about which objects contribute to a Data Bubble. Thus, to solve the lost objects problem, we need to classify the original objects to the closest Data Bubble.

Finally, in step 5 we replace each Data Bubble by the sets of original objects classified to it in a similar way as we did for the weighted versions of our algorithms in section 5. The only difference is that we make use of the virtual reachabilities instead of using the *reachDist* values of the Data Bubbles. We read each object o_i and its classification information from the original database. Let o_i be classified to s_j and B_j be the Data Bubble corresponding to s_j. Now we set the position of o_i to the position of B_j. If o_i is the first object classified to s_j, we set the *reachDist* of o_i to the *reachDist* of B_j, otherwise we set the *reachDist* to *virtual-reachability*(B). Then we write o_i back to disc. Thus, we make one sequential pass (reading and writing) over the original database. As the last action in step 5, we sort the file according to the positions of the objects to generate the final cluster ordering.

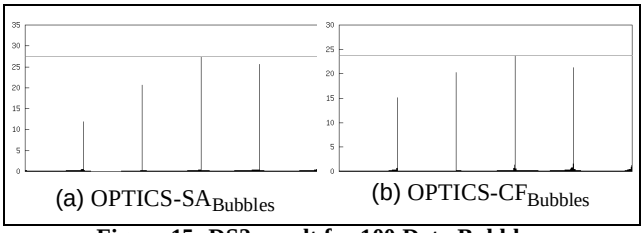

Figure 15: DS2-result for 100 Data Bubbles.

Figure 14(a) shows the results of OPTICS-$SA_{Bubbles}$ for DS1 using sample sizes k=10,000, k=1,000, and k=200. Figure 14(b) shows the same information for OPTICS-$CF_{Bubbles}$. Both algorithms exhibit very good quality for large and medium numbers of Data Bubbles. For very small values of k, the quality of OPTICS-$CF_{Bubbles}$ begins to suffer. The reason for this is the heuristics for increasing the threshold value in the implementation of BIRCH. In phase 2, when compressing the CF-tree down to the maximal number of CFs k, the last increase in the threshold value is chosen too large. Thus, BIRCH generates only 75 Data Bubbles, while sampling produced exactly 200.

Figure 15 shows the results for DS2 and 100 Data Bubbles in which case both algorithms produce excellent results.

Obviously, both OPTICS-$SA_{Bubbles}$ and OPTICS-$CF_{Bubbles}$ solve all three problems (size distortions, structural distortions and lost objects) for high compression rates. OPTICS-$SA_{Bubbles}$ scales slightly better to extremely high compression rates.

9. DETAILED EXPERIMENTAL EVALUATION

In this section, we will discuss both the runtime and the quality issues incurred by compressing the original database into Data Bubbles, and compare them to the original implementation of OPTICS. All experiments were performed on a Pentium III workstation with 450 MHz containing 256MB of main memory and running Linux. All algorithms are implemented in Java and were executed on the Java Virtual Machine Version. 1.3.0beta from Sun. We used approximately 20 GB of space on a local hard disc.

9.1 Runtime Comparison

Runtime and Speed-Up w.r.t. Compression Factor

In figure 16, we see the runtime and the speed-up factors for the database DS1 for different compression factors. Recall that DS1 contains 1 million objects. We used compressions factors of 100, 200, 1,000 and 5,000, corresponding to 10,000, 5,000, 1,000 and 200 representative objects, respectively. Both algorithms are very fast, especially for higher compression rates, with speed-up factors of up to 151 for OPTICS-$SA_{Bubbles}$ and 25 for OPTICS-$CF_{Bubbles}$. Furthermore, we can observe that OPTICS-$SA_{Bubbles}$ is by a factor of 5.0 to 7.4 faster than OPTICS-$CF_{Bubbles}$.

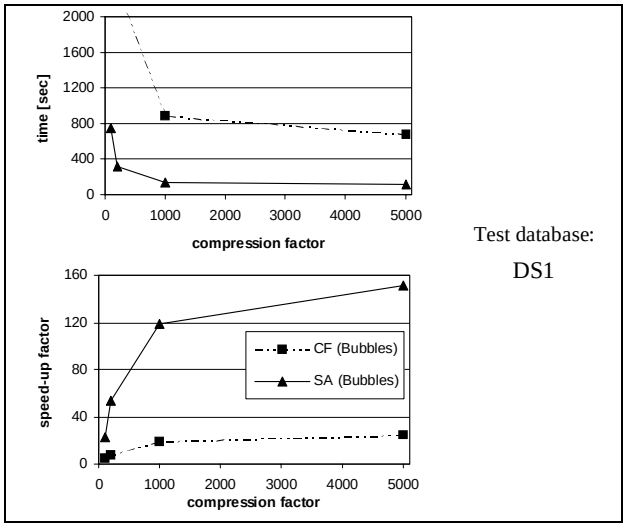

Figure 16: Runtime and speed-up w.r.t. compression factor.

Runtime and Speed-Up w.r.t. the Database Size

Figure 17 shows the runtime and the speed-up factors obtained for different sized databases. The databases were random subsets of DS1. All algorithms scale approximately linearly with the size of the database. An important observation is that the speed-up factor as compared to the original OPTICS algorithm becomes larger (up to 119 for OPTICS-$SA_{Bubbles}$ and 19 for OPTICS-$CF_{Bubbles}$) as the size of the database increases. This is, however, to be expected for a constant number of representative objects, i.e. using one of the proposed methods, we can scale hierarchical cluster ordering by more than a constant factor. Again, OPTICS-$SA_{Bubbles}$ outperforms OPTICS-$CF_{Bubbles}$, by a factor of 6.3 to 8.6.

Runtime and Speed-Up w.r.t. the Dimension

To investigate the behavior of the algorithms when increasing the dimension of the data set, we generated synthetic databases containing 1 million objects in 15 Gaussian clusters of random locations and random size. The databases were generated such that the 10-dim data set is equal to the 20-dim data set projected onto the

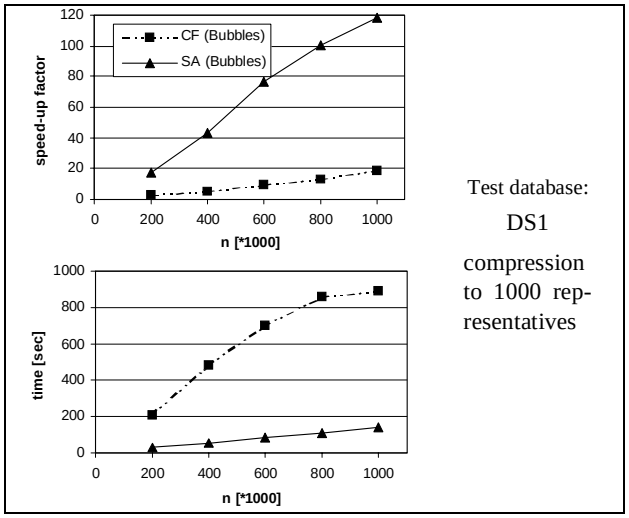

Figure 17: Runtime and speed-up w.r.t. database size.

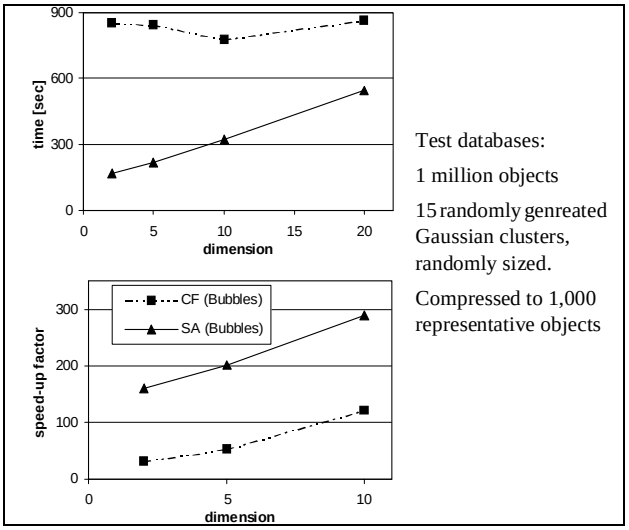

Figure 18: Runtime w.r.t. the dimension of the database.

first 10 dimensions, and the 5-dim is the 10-dim projected onto the first 5 dimensions. Figure 18 shows the runtime and the speed-up factors for these databases. OPTICS-$SA_{Bubbles}$ scales linearly with the dimension of the database, OPTICS-$CF_{Bubbles}$ also contains a linear factor, which is offset by the decreasing number of CFs generated. The speed-up factor for 20-dimensions is not shown because we were unable to run the original algorithm due to main memory constraints. For OPTICS-$SA_{Bubbles}$, the speedup increases from 160 for 2-dimensional databases to 289 for 10-dimensional databases, for OPTICS-$CF_{Bubbles}$ from 31 to 121.

To understand, why BIRCH generated fewer CFs with increasing number of dimensions, recall that BIRCH builds a CF-tree containing CFs in two phases. In phase 1, the original data objects are inserted one by one into the CF-tree. The CF-tree is a main memory structure. This implies that the maximal number of entries in the CF-tree is bounded by main memory. To do this, BIRCH maintains a threshold value. When adding an object to the CF-tree, BIRCH finds the CF-entry which is closest to the new objects. If adding the object to this CF-entry violates the threshold value, the CF-tree is rebuilt by increasing the threshold value and re-inserting all CF-entries into a new CF-tree. Once all original objects are in the CF-tree, BIRCH reduces the number of CFs to a given maximum in phase 2. The CF-tree is repeatedly rebuilt by increasing the threshold value and re-inserting all CFs into a new tree until such a new tree contains no more than the maximal allowed number of CFs. BIRCH uses heuristics to compute the increase in the threshold value. For higher dimensions, this increase is higher, and fewer CFs are generated (429 in the 2-dimensional case, 371 for 5 dimensions, 267 for 10 dimensions and only 16 for the 20-dimensional data set). We used the original heuristics of BIRCH, although, it may be possible to improve the heuristics and thereby solve this problem.

9.2 Quality Evaluation

In the previous sections, we evaluated the quality of the different methods with respect to the compression factor (*c.f.* figure 14 and figure 15) by comparing the different reachability plots. But are the objects in the clusters really *the same* objects as in the original plot? In this subsection we take a closer look at this question, and we will also investigate the scalability of the methods with respect to the dimension of the data.

bs	noise	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
noise	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	40702	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	50	0	69395	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	69174	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	79242	0	0	0	0	0	0	0	0	0	0	0
12	1	0	0	0	0	126617	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	45875	0	0	0	0	0	0	0	0	0
4	7	0	0	0	0	0	0	63198	0	0	0	0	0	0	0	0
7	57	0	0	0	0	0	0	0	93313	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	50318	0	0	0	0	0	0
6	101	0	0	0	0	0	0	0	0	0	65977	0	0	0	0	0
9	113	0	0	0	0	0	0	0	0	0	0	58545	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	38823	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	74603	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14469	0
10	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	109415

Figure 19: Confusion matrix: OPTICS vs. OPTICS-$SA_{Bubbles}$ for a 5-dim database with 1 million objects in 15 Gaussian clusters

Correctness using Confusion Matrixes

To investigate the correctness or accuracy of the methods, we computed a *confusion matrix* for each algorithm. A confusion matrix is a two-dimensional matrix. On one dimension are the cluster ids of the original algorithm and on the other dimension are the ids of the algorithm to validate. We used the 5-dimensional data sets from the previous section, containing 15 Gaussian clusters, which we extracted from the plots. The original algorithm found exactly 15 clusters (with cluster ids 0 to 14). It also found 334 noise objects, i.e. objects not belonging to any cluster.

To compare OPTICS-$SA_{Bubbles}$ and OPTICS-$CF_{Bubbles}$, we compressed the data to 200 objects. Both algorithms found all 15 clusters, which corresponded exactly with the original clusters. The original noise objects are distributed over all clusters. Since the confusion matrixes are in fact identical, we present only the matrix for OPTICS-$SA_{Bubbles}$ in figure 19 (due to space limitations). From left to right we show the clusters found in the original reachability-plot of OPTICS. From top to bottom, we show the clusters found by the OPTICS-$SA_{Bubbles}$. The rows are reordered so that the largest numbers are on the diagonal.

Quality w.r.t. the Dimension of the Database

Figure 20 shows the reachability-plots for the different dimensional databases, which we already used for the runtime experiments. Both algorithms find all 15 clusters with the correct sizes, with the OPTICS-$SA_{Bubbles}$ being about twice as fast as OPTICS-$CF_{Bubbles}$. Also, the quality of OPTICS-$SA_{Bubbles}$ is slightly better: it shows the gaussian shape of the clusters, and OPTICS-$CF_{Bubbles}$ does not.

9.3 Real-World Database

To evaluate our compression techniques on a real-world database, we used the Color Moments from the Corel Image Features available from the UCI KDD Archive at kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html. This database contains image features (color moments) extracted from a Corel image collection. We used the first order moments in the HSV color scheme, as the Euclidean distance can be used to measure distance in this feature database containing 68,040 images.Figure 21 (a) shows the result of OPTICS on the whole data set. This data set is particularly challenging for a clustering algorithm using data compression because in this setting it contains no significant clustering structure, apart from two very small clusters, i.e. the two tiny clusters are embedded in an area of lower, almost uniform density.

For OPTICS-$CF_{Bubbles}$ and OPTICS-$SA_{Bubbles}$ we used 1,000 representative objects, i.e. a compression by a factor of 68. The runtime of OPTICS was 4,562sec, OPTICS-$CF_{Bubbles}$ took 76sec and OPTICS-$SA_{Bubbles}$ 20sec to generate the cluster ordering. Thus, the speedup-factors were 60 and 228 respectively. The result of OPTICS-$CF_{Bubbles}$ (which generated only 547 CFs by setting the parameter for the desired number of leaf nodes to 1000) approximates the general structure of the data set, but looses both clusters. The result of OPTICS-$SA_{Bubbles}$ nicely shows the general distribution of the data objects and also recovers both clusters.

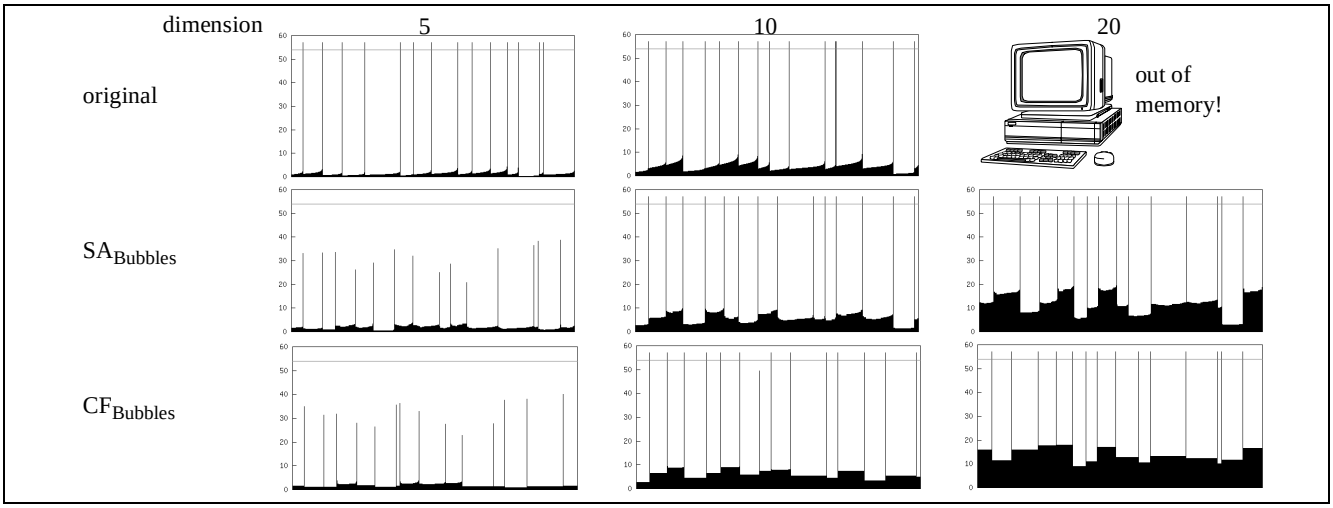

Figure 20: Results for different dimensional databases (1 mio objects, compressed to 200 representatives)

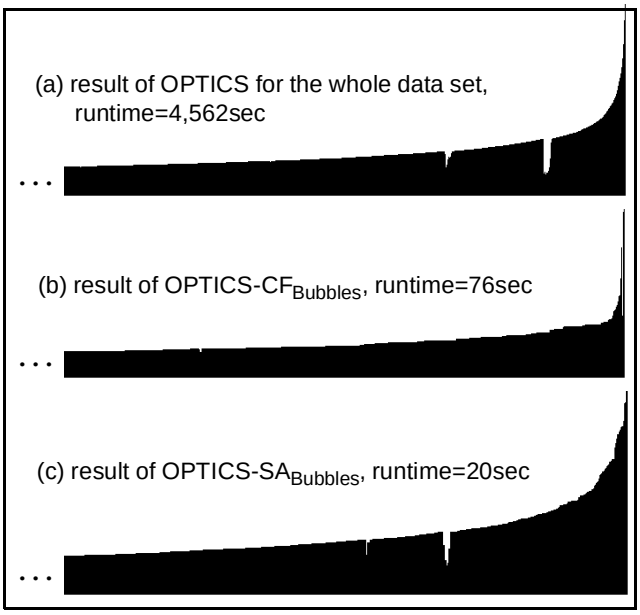

Figure 21: Results for the Corel Image Features database

To validate that the two clusters in fact contain the same objects, we extracted them manually and computed the confusion matrix (cf. figure 22, columns = OPTICS, rows = OPTICS-SA$_{Bubbles}$). The clusters are well-preserved, i.e. no objects switched from one cluster to the other one. Due to the general structure of the database, some of the objects bordering the clusters are assigned to the clusters or not assigned to the clusters, depending on the algorithm used. This example shows, that OPTICS-SA$_{Bubbles}$ can even find very small clusters embedded in a very noisy database.

	noise	0	1
noise	67087	59	19
0	75	253	0
1	20	0	527

Figure 22: Confusion Matrix

9.4 Discussion

If we wish to analyze groups of objects in a very large database, after applying a hierarchical clustering algorithm to compressed data, we must use at least the *weighted* versions of our algorithms, because of the "lost objects problem". We did not include the runtimes of these algorithms in our diagrams because they are almost indistinguishable from the runtimes using Data Bubbles. However, we have seen that the weighted versions work well only for very low compression factors, which results in a much larger runtime as compared to using Data Bubbles - for a result of similar quality.

10. CONCLUSIONS

In this paper, we developed a version of OPTICS using data compression in order to scale OPTICS to extremely large databases. We started with the simple and well-known concept of random sampling and applying OPTICS only to the sample. We compared this with executing the BIRCH algorithm and applying OPTICS to the centers of the generated Clustering Features. Both methods incur serious quality degradations in the result. We identified three key problems: lost objects, size distortions and structural distortions.

Based on our observations, we developed a post-processing step that enables us to recover some of the information lost by sampling or using BIRCH, solving the lost objects and size distortions problems. This step classifies the original objects according to the closest representative and replaces the representatives in the cluster ordering by the corresponding sets of original objects.

In order to solve the structural distortions, we introduced the general concept of a *Data Bubble* as a more specialized kind of compressed data items, suitable for hierarchical clustering. For Euclidean vector data we presented two ways of generating Data Bubbles efficiently, either by using sampling plus a nearest neighbor classification or by utilizing BIRCH. We performed an experimental evaluation showing that our method is efficient and effective in the sense that we achieve high quality clustering results for data sets containing hundred thousands of vectors in a few minutes.

In the future, we will investigate methods to efficiently generate Data Bubble from non-Euclidean data, i.e. data for which only a distance metric is defined. In this setting, we can no longer use a method such as BIRCH to generate sufficient statistics, but we can still apply sampling plus nearest neighbor classification to produce data sets which can in principle be represented by Data Bubbles. The challenge, however, is then to *efficiently* determine a good representative, the radius and the average *k*-nearest neighbor distances needed to represent a set of objects by a Data Bubble.

References

[1] Ankerst M., Breunig M. M., Kriegel H.-P., Sander J.: "*OPTICS: Ordering Points To Identify the Clustering Structure*", Proc. ACM SIGMOD Int. Conf. on Management of Data, Philadelphia, PA, 1999, pp 49-60.

[2] Bradley P. S., Fayyad U., Reina C.: "*Scaling Clustering Algorithms to Large Databases*", Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, New York, NY, AAAI Press, 1998, pp. 9-15.

[3] Breunig M., Kriegel H.-P., Sander J.: "*Fast Hierarchical Clustering Based on Compressed Data and OPTICS*", Proc. 4th European Conf. on Principles and Practice of Knowledge Discovery in Databases, LNCS Vol. 1910, Springer Verlag, Berlin, 2000, pp. 232-242.

[4] DuMouchel W., Volinsky C., Johnson T., Cortez C., Pregibon D.: "*Sqashing Flat Files Flatter*", Proc. 5th Int. Conf. on Knowledge Discovery and Data Mining, San Diego, CA, AAAI Press, 1999, pp. 6-15.

[5] Ester M., Kriegel H.-P., Sander J., Xu X.: *"A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise"*, Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, 1996, pp. 226-231.

[6] Jain A. K. and Dubes R. C.: "*Algorithms for Clustering Data*", Prentice-Hall, Inc., 1988.

[7] Kaufman L., Rousseeuw P. J.: *"Finding Groups in Data: An Introduction to Cluster Analysis"*, John Wiley & Sons, 1990.

[8] MacQueen J.: *"Some Methods for Classification and Analysis of Multivariate Observations"*, Proc. 5th Berkeley Symp. Math. Statist. Prob., 1967, Vol. 1, pp. 281-297.

[9] Sibson R.: "*SLINK: an optimally efficient algorithm for the single-link cluster method*", the Computer Journal Vol. 16, No. 1, 1973, pp. 30-34.

[10] Zhang T., Ramakrishnan R., Linvy M.: *"BIRCH: An Efficient Data Clustering Method for Very Large Databases"*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Montreal, Canada, ACM Press, New York, 1996, pp. 103-114.