Data Bubbles for Non-Vector Data: Speeding-up Hierarchical Clustering in Arbitrary Metric Spaces

Jianjun Zhou Jörg Sander

University of Alberta, Department of Computing Science
Edmonton, Alberta, Canada T6G 2E8
{ jianjun, joerg }@cs.ualberta.ca

Abstract

To speed-up clustering algorithms, data summarization methods have been proposed, which first summarize the data set by computing suitable representative objects. Then, a clustering algorithm is applied to these representatives only, and a clustering structure for the whole data set is derived, based on the result for the representatives. Most previous methods are, however, limited in their application domain. They are in general based on sufficient statistics such as the linear sum of a set of points, which assumes that the data is from a *vector space*. On the other hand, in many important applications, the data is from a metric *non-vector space*, and only distances between objects can be exploited to construct effective data summarizations. In this paper, we develop a new data summarization method based only on distance information that can be applied directly to non-vector data. An extensive performance evaluation shows that our method is very effective in finding the hierarchical clustering structure of non-vector data using only a very small number of data summarizations, thus resulting in a large reduction of runtime while trading only very little clustering quality.

1. Introduction

Data Clustering is an important task for knowledge discovery in databases (KDD). The basic goal of a clustering algorithm is to partition a set of data objects into groups so that similar objects belong to the same group and dissimilar objects belong to different groups. There are different types of clustering algorithms for different types of applications. A common distinction is between *partitioning* and *hierarchical* clustering algorithms (see e.g. [9]). Partitioning algorithms are, for instance, the *k*-means [10] and the *k*-medoids algorithms [9]. Partitioning algorithms decompose a database into a set of *k* clusters whereas hierarchical algorithms only compute a representation of the data set, which reflects its hierarchical clustering structure, but do not explicitly determine clusters. Examples of hierarchical clustering algorithms are the Single-Link method [11] and OPTICS [1].

Clustering algorithms in general, and in particular hierarchical algorithms, do not scale well with the size of the data set. On the other hand, very fast methods are most desirable for exploratory data analysis, which is what clustering is mostly used for.

To speed-up cluster analysis on large data sets, some data summarization methods have been proposed recently. Those methods are based on a general strategy that can be used to scale-up whole classes of clustering algorithms (rather than inventing a new clustering algorithm):

1) Use a data summarization method that produces “sufficient statistics” for subsets of the data set (using either sampling plus a classification of objects to the closest sample point, or some other technique such as BIRCH [12]). The data summarizations are sometimes also called “micro-clusters” (e.g. in [8]).
2) Apply (an adapted version of) the clustering algorithm to the data summaries only.
3) Extrapolate from the clustering result for the data summaries a clustering result for the whole data set.

Different data summarization methods have different advantages and disadvantages. In [3] it was shown that hierarchical clustering algorithms such as the Single-Link [11] method or OPTICS [1] require special information in order to produce high quality results for small numbers of data summaries. The proposed data summarizations that meet all the requirements for hierarchical clustering were called “Data Bubbles”.

Most techniques to compute data summaries, including Data Bubbles, are based on the assumption that the data is from a vector space. Typically, they compute sta-

Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003

tistics such as the mean of the set of objects which requires that vector space operations (addition of objects, multiplication with scalar values) can be applied.

For non-vector spaces, the only information that can be utilized is a similarity or a dissimilarity distance function. In this paper we will assume a distance function to measure the dissimilarities, i.e., we only have information about distances between objects. This makes it difficult or at least very expensive to compute the usual sufficient statistics used to summarize vector data. However, having a data summarization method that allows a very fast (even if only approximate) clustering of non-vector data is highly desirable since the distance functions for some typical and important applications can be extremely computationally expensive (e.g., a sequence alignment for a set of RNA or amino acid sequences).

In this paper, we propose a novel data summarization method that can be applied to non-vector data to produce high-quality "micro-clusters" to efficiently and effectively support hierarchical clustering. The information produced for each data summary is related and improves upon the information computed for the Data Bubbles proposed in [3] in the sense that accurate estimations of the information needed by hierarchical clustering algorithms is generated (in fact, we suggest to use our new version of Data Bubbles even for vector data).

The rest of the paper is organized as follows. We briefly review related work in section 2. We present the necessary background regarding the original Data Bubbles for vector data and the clustering algorithm OPTICS in section 3. Section 4 discusses the problems when trying to generate summary information for sets of non-vector data and introduces our new method. The experimental evaluation in section 5 shows that our method allows not only very effective and efficient hierarchical clustering of non-vector data, but also that it even outperforms the original Data Bubbles when applied to vector data. Section 6 concludes the paper.

2. Related Work

The most basic method to speed-up expensive data mining algorithms such as hierarchical clustering is probably random sampling: only a subset of the database is randomly chosen, and the data mining algorithm is applied to this subset instead of to the whole database. Typically, if 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 to support, in particular, clustering algorithms. For *k*-means type of clustering algorithms, a summary statistics called "clustering features", originally introduced for the Birch method [12], has been used by different approaches. Birch incrementally computes compact descriptions of subclusters, called Clustering Features, which are defined as $CF = (n, LS, ss)$, where LS is the linear sum and ss the square sum of the n points in the sub-cluster represented by the clustering feature CF.

The *CF*-values are sufficient to compute information like centroid, radius and diameter of a set of points. They also satisfy an additivity condition, that allows the incremental computation of *CF*-values when inserting points into a set: 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 union of the points in S_1 and S_2, $S_1 \cup S_2$.

The *CF*s are organized in a balanced tree with branching factor B and a threshold T, where a non-leaf node represents all objects in the whole sub-tree that is rooted at this node. 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. The generation of a *CF*-tree is similar to the construction of B^+-trees: points p are inserted into the tree by finding first the leaf in the current *CF*-tree that is closest to p. If an entry in the leaf can absorb p without violating the threshold condition, it is inserted into this entry and the corresponding CF value is updated. If p cannot be inserted into an existing entry, a new entry is created in the leaf node. This may lead to an overflow of the leaf node causing it (and possibly its ancestors) to be split in a similar fashion as B-trees. A clustering algorithm is then applied to the entries in the leaf nodes of the *CF*-tree.

In [2], a very specialized compression technique for scaling-up *k*-means and EM clustering algorithms is proposed. This method basically uses the same type of sufficient statistics as Birch, i.e. triples of the form (n, LS, ss). The major difference is only that different sets of data items are treated and summarized independently: points that are unlikely to change cluster membership in the iterations of the clustering algorithm, data summaries that represent tight sub-clusters of data points, and a set of regular data points which contains all points which cannot be assigned to other data summarizations.

In [4], a general framework for "squashing" data is proposed, which is intended to scale up a large collection of data mining methods. The method is based on partitioning the dimensions of the data space and grouping the points into the resulting regions. For each region, a number of moments are calculated such as mean, minimum, maximum, second order moments such as X_i^2 or X_iX_j, and higher order moments depending on the desired degree of approximation. Squashed data items are then created for each region in a way that the moments of the squashed items approximate those of the original data falling into the region. This information can also be used to compute clustering features as above for each constructed region in order to speed-up *k*-means type of clustering algorithms.

In [3] it was also proposed to compute sufficient statistics of the form (n, LS, ss) based on a random sample by partitioning the data set using a *k*-nearest neighbour classification. This method has several advantages over, for instance the CF-tree: the number of representative objects for a data set can be determined exactly, and no other heuristic parameters such as a maximum diameter, or a bin-size have to be used in order to restrict the number of partitions that are represented by triples (n, LS, ss). The method was proposed as follows:

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

Our experiences in [3] showed that the quality of the sufficient statistics obtained by random sampling is much better than the *CF*-values produced by Birch, when used to generate the additional information that is needed to get satisfactory results with hierarchical clustering algorithms. The runtime to generate those CF values using a CF-tree is also significantly larger and make it almost impossible to beat even a naïve sampling approach to speed-up clustering, given the same resources. If it takes too much time to generate data summarizations, naïve sampling may just use a larger sample and obtain superior results with a much less complex implementation.

The only other proposal for a data summarization method for *non-vector* data that we are aware of is presented in [6], and is based on Birch. The authors suggest a generalization of a Birch tree that has two instances BUBBLE and BUBBLE-FM for non-vector data. Both methods keep a number of representatives for each leaf node entry in order to approximate the most centrally located object in a CF-tree leaf. In non-leaf level entries, both methods keep a certain number of sample objects from the sub-tree rooted at that entry in order to guide the search process when building the tree. The basic difference between BUBBLE and BUBBLE-FM is that for BUBBLE-FM the sample points in the non-leaf node entries are mapped to a d-dimensional Euclidean vector space using Fastmap [5]. The image space is then used to determine distances between new objects and the CFs, thus replacing possibly expensive distance calculations in the original space by Euclidean distance computations. We will argue that this approach has similar drawbacks as the vector version, and we will therefore, base our current work for non-vector data on a sampling based approach to produce data summarizations.

3. Data Bubbles for Euclidean Vector Data

In this section, we briefly review the notion of Data Bubbles for Euclidean vector spaces as proposed in [3]. We discuss the special requirements that *hierarchical* clustering algorithms such as the Single-Link method and OPTICS pose on data summarization methods, and we illustrate the advantages of Data Bubbles.

While simple statistics such as clustering features produced by Birch, are effective for k-means type clustering algorithms, they typically are not sufficient to produce good results using a hierarchical clustering algorithm. The main reason is that hierarchical clustering algorithms are based on the distances between sets of data points which are not represented well by the distances between only the representative objects, especially when the compression rate increases. This type of error typically results in a very distorted clustering structure based on data summaries. The Data Bubbles in [3] have been proposed to solve those problems, showing that a data summarization method, in order to support hierarchical clustering, has to take into account the extension and the point density of the data-subset being represented.

3.1 Basic Definitions

A Data Bubble was defined in [3] as follows:

Definition 3.1: A *Data Bubble* for a set of points $X=\{X_i\}$, $1 \le i \le n$, is a tuple $B_X = (rep, n, extent, nnDist)$, where

- *rep* is a representative object for X, which is assumed to be close to the centre of the set X;
- n is the number of objects in X;
- *extent* is the radius of B_X around *rep* that encloses "most" of the objects in X;
- $nnDist(k,B_X)$ is a function that estimates the average k-nearest-neighbour distances in B_X.

For d-dimensional points from a Euclidean vector data, the representative *rep*, the radius of the Data Bubbles *extent*, and the k-nearest-neighbour distances $nnDist(k, B)$ can be easily estimated using simple sufficient statistics, which can be incrementally computed during the initial construction of the Data Bubbles.

The representative *rep* is simply computed as the mean of the set of objects in X, i.e., $rep = \left(\sum_{i=1\ldots n} X_i\right) / n$.

The radius of B_X around *rep* can be estimated by the average pair-wise distances within B_X, i.e., $extent = \sqrt{\frac{\sum_{i=1\ldots n}\sum_{j=1\ldots n}(X_i - X_j)^2}{n\cdot(n-1)}}$. This expression can in turn be computed from the simpler statistics linear sum LS and square sum ss of all objects in X. LS and ss can be incrementally maintained when constructing a Data Bubble (as in the construction of cluster features CF in the BIRCH algorithm). Using these two values, the *extent* can be calculated as $\sqrt{\frac{2\cdot n\cdot ss - 2\cdot LS^2}{n\cdot(n-1)}}$.

The average k-nearest-neighbour distances can be estimated by a simple arithmetic expression if a uniform distribution of objects within a Data Bubble is assumed. This assumption is quite robust in many applications since a Data Bubble only represents a small fraction of a data set and the uniformity assumption holds approximately; under this assumption: $nnDist(k,B_X) = \left(\frac{k}{n}\right)^{\frac{1}{d}} \cdot extent$.

3.2 Application to Hierarchical Clustering

Hierarchical clustering algorithms compute a hierarchical representation of the data set, which reflects its possibly nested clustering structure.

The algorithm OPTICS is based on the notions of core-distance and reachability-distance with respect to parameters *Eps* and *MinPts*. The core-distance of a point p represents the point density around p; the reachability distance is a "smoothed" measure of the distance between points that avoids Single-Link effects. Using these distances, OPTICS computes a "walk" through the data set, and assigns to each object p its core-distance and the smallest reachability-distance *reachDist* with respect to an object considered before p in the walk. The algorithm starts with an arbitrary object assigning it a reachability-distance equal to ∞. The next object o in the output is then always the object that has the shortest reachability distance d to *any* of the objects that were "visited" previously by the algorithm. This reachability-value d is assigned to this object o. The output of the algorithms is a *reachability plot*, which is a bar plot of the reachability values assigned to the objects in the order they were visited. An example reachability plot for a 2-dimensional data set is depicted in Figure 1. Such a plot is interpreted as following: "valleys" in the plot represent clusters, and the deeper the "valley", the denser the cluster. The tallest bar between two "valleys" is a lower bound on the distance between the two clusters. Large bars in the plot, not at the border of a cluster represent noise, and "nested valleys" represent hierarchically nested clusters.

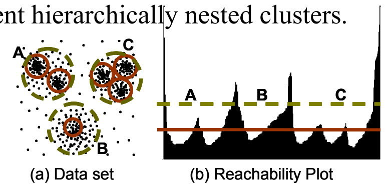


Figure 1. Example reachability plot and dendrogram

Clusters in a hierarchical clustering representation are in general obtained manually (e.g., by cutting through the representation). This process is typically guided by a visual inspection of the diagram – which is why a correct representation of the clustering structure is very important, especially when applying the algorithm to data summarizations instead of the whole data set.

The most important issue when applying hierarchical clustering to Data Bubbles is the distance function that is used to measure dissimilarity between two Data Bubbles. In [3] it has been shown that using the distance between representatives and the extent of Data Bubbles for vector data, a distance between Data Bubbles can be computed that dramatically improves the result of hierarchical clustering compared to using only the distance between representatives. This notion of distance that is aware of the extent of Data Bubbles is depicted in Figure 2. If the Data Bubbles do not overlap, it is basically the distance between the "borders" of the Data Bubbles (distance between representative objects of the Data Bubbles minus the extents of the Data Bubbles plus the 1-nearest neighbour distances of the Data Bubbles), otherwise, i.e., if they overlap, it is the estimated nearest neighbour distance of the Data Bubble that has the larger *nn*-distance.

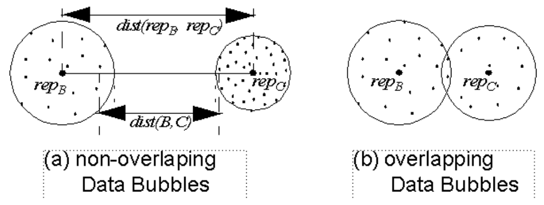


Figure 2. Illustration of the distance between original Data Bubbles for vector data (Fig. adapted from [3])

The second important issue in hierarchical clustering of data summarizations is the adaptation of the graphical result. The reason is that the Data Bubbles typically represent sets of objects that may contain significantly different numbers of objects, and that can have largely differing point densities. Including only the representative of a Data Bubble in the hierarchical output representation will most often lead to a very distorted picture of the true clustering structure of a data set. Therefore, for OPTICS, the bar for each Data Bubble in the reachability plot is expanded using the so-called "virtual reachability". More precisely, for each Data Bubble representing n points, n bars are added to the reachability plot. The height of each bar is calculated as the virtual reachability of the Data Bubble, which corresponds to the estimated average reachability distance for points within the Data Bubble (basically the estimated *MinPts*-nearest neighbour distance). Other hierarchical algorithms such as the Single-Link method can be similarly adapted to work with Data Bubbles. Due to page limitations, we cannot discuss these extensions here.

4. Data Bubbles for Non-Vector Spaces

The only information that can be utilized in a non-vector space is the distance function, i.e., information about distances between objects. Therefore, it is difficult in such spaces to get an accurate and at the same time computationally inexpensive estimation for the important components defined for the original Data Bubbles. We cannot compute new "artificial" objects such as a mean, which is guaranteed to be in the centre of the respective set, and hence would be the best representative for the objects in a Data Bubble. We also cannot compute statistics like the linear sum or the square sum of the objects that would allow us to incrementally maintain a good estimation of the radius of a Data Bubble around the representative. Similarly, there is no inexpensive or incremental way to compute an estimation of the average k-nearest-neighbour distances in a Data Bubble. We will argue that for these reasons the original definition of Data Bubbles has to be significantly changed and adapted in order to deal with the particular problems of non-vector spaces.

The main purpose of Data Bubbles is to support effective and highly efficient hierarchical clustering based on the summary information provided by the Data Bubbles. The representative, the extent, and the average k-nearest-neighbour distances of a Data Bubble serve only the purpose of defining effective distance notions for hierarchical clustering. For the algorithm OPTICS, which we will use to evaluate our method, these notions are:

- The notion of a distance between Data Bubbles, which has to "be aware" of the extension of the Data Bubbles. This is the most important notion for effective hierarchical clustering.
- The core-distance of a Data Bubble, which is also used to define the "virtual reachability" for objects, needed to correctly expand a reachability plot.
- The reachability-distance of a Data Bubble relative to another Data Bubble, which is needed during the execution of OPTICS. The appropriateness of the reachability-distance is dependent on the previous two notions, since it is defined using only core-distance and the distance between Data Bubbles.

Errors in estimating a representative, the extent, or the average k-nearest-neighbour distances will lead to errors when computing the above distances, which in turn will produce errors in the clustering result using Data Bubbles. To make things worse: errors for different components in a Data Bubble may depend on and amplify each other, e.g., an error in the estimation of the representative will obviously lead to an increased error in the extent around the representative, if we keep the original definition of extent as a radius around the representative that contains most of the objects of the Data Bubble.

In the following sub-sections we will analyze these problems and propose a new and more suitable version of Data Bubbles that solves these problems. In order to discuss the problems, we will assume the following minimal procedure to generate s Data Bubbles for non-vector data (the complete method will be given later in this section):

1. Sample s objects from the database randomly.
2. Assign each object in the database to the closest sample object from the set of objects obtained in step 1.

This means that using this procedure, the only information we can utilize in our computation of data summarizations are the s objects drawn from the database in step 1 (they may be used, for instance, as candidates for representative objects), and the distances of all objects to all the sample objects obtained in step one. These distances have to be computed in any case to determine the closest representative for each object.

4.1 Representative Objects

In a non-vector space the representative object for a Data Bubble has to be an object from the Data Bubble itself since we cannot compute a mean for the set of objects. Theoretically, the best representative for a set of objects in a non-vector space is a *medoid*, i.e. an object that is located most centrally in the set of objects, in the sense that its overall distance to all other objects in the Data Bubble is minimal. More formally:

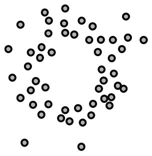
Definition 4.1 A *medoid* for a set of objects X is an object $m \in X$ such that for all $p \in X$: $\sum_{o \in X} dist(m,o) \leq \sum_{o \in X} dist(p,o)$.

A medoid, although it seems to be the best choice of a representative has a severe drawback: determining a medoid for a set of n objects is computationally expensive ($O(n^2)$), since all pair-wise distances have to be computed. Because we want to use very high compression rates in practice (i.e., only a very small number of Data Bubbles, and hence a very large number of objects represented by one Data Bubble on average), it is not feasible to determine a medoid for a Data Bubble with this exhaustive search method. The same amount of computation could be better used to cluster a larger subset of objects directly without generating Data Bubbles.

Using our minimal procedure to construct data summarizations, there are basically three alternatives to determine some representative objects for a Data Bubble more efficiently but also less optimally – all with advantages and disadvantages:

1. "*Initial sample object*": keep the initial sample object that is used to generate a Data Bubble as the representative of the Data Bubble.
2. "*Relocation using a sub-sample*": after the generation of the Data Bubble, take a small sub-sample from the Data Bubble, including the initial sample object, and determine an exact medoid only in this subset.
3. "*Maintaining several candidates*": while generating the Data Bubble, keep a number of objects as potential representatives in main memory (e.g. first m objects assigned to the Data Bubble). When assigning objects, compute and sum up distances not only to the initial sample object but also to the additional candidates in the Data Bubble. After the generation of the Data Bubble, select the candidate with the lowest sum of distances.

The first alternative, keeping the initial sample object, is the least expensive, since no additional computation is necessary. But, it is also the alternative with the largest error. The quality of the representatives found by the second alternative obviously depends on the size of the sub-sample drawn from the Data Bubble. Our experiments showed however, that for instance a 5% sub-sample of a Data Bubble will result in representatives that are only slightly better approximations of a true medoid, and the effect on the quality of the clustering result is not significant. Taking larger sub-samples, however, is also too expensive in the same sense as the exhaustive method: instead of taking a sub-sample to relocate the representative, we can use a larger sample without bubble generation to improve the clustering result. Alternative 3, i.e., maintaining several candidates during the generation of the bubbles, has basically the same properties as alternative 2.



Note that, even in the best case, i.e., if we could get an exact medoid for the whole set of objects in a Data Bubble, we may produce noticeable errors in the clustering result because there are limits to the accuracy of a medoid as being in the centre of the set of objects that it represents. This is in fact a drawback for any representative object that has to be an element of the set itself (opposed to a computed mean in case of vector data). The figure to the left depicts an extreme case where the data set does not contain an object close to the centre of the set. Due to this drawback, even the best selection of a representative for a Data Bubble may result in an error when estimating the extent of a Data Bubble and consequently in the distance between Data Bubbles to a degree that would not occur for vector Data Bubbles.

Using any of the three alternatives, and keeping the original definition of a Data Bubble, we cannot guarantee that our representative will be close enough to the "centre" of the data set to avoid errors. On the other hand, having a representative close to the centre of a Data Bubble is not an objective in its own for hierarchical clustering. Only the above listed distance notions for Data Bubbles are important. As we will see in the next subsection, we can in fact compensate for a less centred representative by applying a new and much more sophisticated distance function for Data Bubbles. Representatives that are not close to the centre of a data set will only lead to an error in the clustering result when using the original idea of extent of a Data Bubble around a representative and the original definition of distance that is based on this extent.

Therefore, we choose alternative 1 and keep the initial sample object as the representative of a non-vector Data Bubble, which has no computational overhead.

4.2 Average *knn*-Distances, Core-Distance, and Virtual Reachability Distance

The estimation of the average *k*-nearest-neighbour distances *nnDist*(*k*, *B*) for a Data Bubble *B* is closely related to the core-distance and the virtual reachability distance of *B*. The nearest neighbour distance is also used in the original definition of the distance between Data Bubbles.

Because there is no notion of volume and dimensionality in a non-vector space, we cannot apply a simple function to calculate the average *k*-nearest-neighbour distances as in a vector space. When constructing Data Bubbles for non-vector data, we have similar alternatives to determine an estimation of the average *k*-nearest-neighbour distances as we have for the selection of a representative object (using a sub-sample of the objects in a Data Bubble and compute the *k*-nearest-neighbour distances only in this sub-sample is, however, not an option: they would very likely be highly overestimated):

1. "*knn-distances w.r.t. the initial sample object*": when assigning objects to Data Bubbles, maintain a list of the *k* smallest distances relative to each initial sample object. For each Data Bubble, simply use the k smallest-distances to its representative as the estimation of the average *knn*-distance in the whole Data Bubble.
2. "*knn-distances w.r.t. several reference objects*": keep a number of objects from a Data Bubble in main memory (e.g. the first *m* objects assigned to the Bubble) and compute distances to these objects for all objects that are assigned to the Data Bubble. For each of the reference objects, maintain a list of the *k* smallest distances. After the generation of the Data Bubble, compute an estimation of the average *k*-nearest-neighbour distances by averaging those values.

As for the selection of the representative objects, the first alternative has no significant computational overhead since the distances to the initial sample objects have to be computed anyway. The computational cost and the improvement in the estimation of the *knn*-distances for the second alternative depend on the number of reference object that are kept in main memory. As before, if we keep too many reference objects, the gain in accuracy will not outweigh the increased number of distance computations. For the same amount of additional distance computations, we may be able to get a better result by just taking a larger sample size to begin with.

The important question regarding the average *knn*-distances in a Data Bubble is: for which values of *k* do we need the estimation and how accurate do they have to be? The most important use of the *knn*-distances is for estimating the core-distance of a Data Bubble. The core-distance also defines the virtual reachability value for a Data Bubble, which is used when "expanding" a Data Bubble in the clustering output. Typically, we don't want to use values for *MinPts* that are too small, in order to avoid single-link effects and to reduce the effect of noise (see [1] for details). In practice, we mostly use values which are significantly larger than 5 for larger data sets; and we may consider *MinPts* values in the range of 5 only for relatively small data sets. To estimate the core- and virtual reachability distance of a Data Bubble we therefore only need *knn*-distances for the larger values of *k*=*MinPts* that we want to use for clustering. Fortunately, the larger values of the average *knn*-distance in a Data Bubble can be estimated with an acceptable error using only the distances to the initial sample object. In fact, the larger *k*, the more accurate the estimation using only the initial sample object (or any other reference object, or the average over several reference objects). Only for very small values of *k*, especially for *k*=1, is the actual value *nnDist*(1, *B*) for most of the objects in *B* quite different from the average nearest-neighbour distance. The nearest neighbour distance in a Data Bubble *B*, *nnDist*(1, *B*), is only used in the original definition of distance between Data Bubbles, which we will not use for non-vector data because of other reasons. Therefore, we don't need the more error prone estimation of *nnDist*(*k*, *B*) for very small values of *k*. And, since the use of only a few reference objects does not significantly improve the result for the larger values of k, we choose here again the more efficient alternative 1 to estimate the *knn*-distances (up to the maximum value of *MinPts*), i.e., we use only the initial sample objects and the distances that we have to compute in the construction of Data Bubbles anyway.

Using the estimation of k-nearest neighbour distances, the *core-distance* of a Data Bubble B (average distance of objects in B to the *MinPts* nearest neighbour – so that they are core objects), and virtual reachability distance of B (the distance needed for the expansion of the reachability plot after clustering) are then defined similarly as in [3]:

Definition 4.2 Let B be a Data Bubble. The virtual reachability and core-distance of B are defined using the estimated *knn*-distances, *nnDist*(k, B), as following:

$$virtualReachability(B)=core\text{-}dist(B)=nnDist(MinPts,B).$$

4.3 The Distance Between Data Bubbles

The original distance between Data Bubbles in [3] is based on the extents of the Data Bubbles as illustrated above in Figure 2. The purpose of the extent of a Data Bubble is to be able to define the distance between Data Bubbles as the distance between their *borders*, which are approximated by the extents.

However, the extent as the average pair-wise distance in a Data Bubble is expensive to estimate since there is no supporting statistics that could be collected incrementally while constructing a Data Bubble. The only option to get an estimation of the average pair-wise distance would be to draw a sub-sample of objects from a Data Bubble and compute all pair-wise distances in this sub-sample. The accuracy of this approach depends on the size of the sub-sample. The value could be used as a radius around the representative within which most of the objects of the Data Bubble are supposed to be located. Since this is the intended interpretation of the extent, we could alternatively use only the distances to the representative and maintain incrementally a distance around the representative so that "most" objects of the Data Bubble fall inside this radius around the representative (similar to maintaining the *knn*-distances). The second alternative for estimating a global extent is much more efficient but also much more error-prone since it is very sensitive to outliers.

To work properly, both approaches have to assume (in addition to having a small error) that the representative is close to the centre, which we know, we cannot guarantee in a non-vector space. In fact, errors in choosing representatives and errors in the estimation of the extent amplify each other, resulting in large errors in the clustering result, because the distances between Data Bubbles will be heavily distorted. As a solution, we propose a new definition of distance between Data Bubbles, which is based on simple statistics that uses only the distances between objects and sample objects (which are computed when constructing a Data Bubble anyway). All the needed notions can be maintained incrementally and without significant computational overhead.

Conceptually, in order to compensate for an error in the selection of the representatives, we want to distinguish the extent of a Data Bubble around its representative in different *directions* – "*direction*" being defined using only distances between the representative and other objects. For instance, if a representative is not centreed well in a Data Bubble, the distances to the "border" of the Data Bubble may be very different in different "directions". Figure 3 illustrates this concept using a 2-dimensional Data Bubble B where the extent from the representative rB in direction of object O_1 is much smaller than the extent in direction of object O_2. The notions required to formalize these intuitions will be introduced in the following. Please note that all concepts will be defined without any reference to vector space properties or operations, and that although we will use 2-dimensional point data to illustrate the concepts, the notions are solely based on distances between objects.

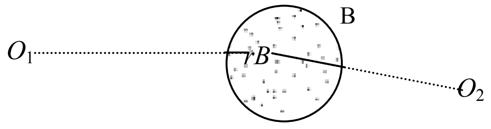


Figure 3. "Directional extent" of a Data Bubble

In order to define more accurate distances between Data Bubbles, the goal is to find a more accurate representation of the "border" of a Data Bubble. However, we only need to know the distance between the representative and the border, i.e. the extent of a Data Bubble, in the directions of the (representatives of) other Data Bubbles. Intuitively, given any two Data Bubbles, A and B, and their representatives, rA and rB, we can divide the Data Bubble B into two parts with respect to Data Bubble A: one part containing the objects in B that are "in the direction of A" in the sense that the distance between them and the representative of A, rA, is smaller than the distance between the two representatives rA and rB; the second part of B contains the other objects, which are called to be "in the reverse direction of A". Formally:

Definition 4.3 Let A and B be two sets of objects, represented by rA and rB, respectively.

- *Bubble*(B).*InDirection*(A):=
  $\{o \in B \mid \text{dist}(o, rA) \le \text{dist}(rA, rB)\}$
  For each object $o \in$ *Bubble*(B).*InDirection*(A) we say that o lies in the direction of A.
- *Bubble*(B).*InRevDirection*(A):=
  $\{o \in B \mid \text{dist}(o, rA) > \text{dist}(rA, rB)\}$
  For each object $o \in$ *Bubble*(B).*InRevDirection*(A) we say that o lies in the reverse direction of A.

Figure 4 illustrates these notions: all objects $o \in B$ that lie inside the circle having rA as centre and the distance between rA and rB as radius, are in direction of A, i.e. in *Bubble*(B).*InDirection*(A); objects $o' \in B$ which are outside the circle lie in "reverse" direction of A, i.e. in *Bubble*(B).*InRevDirection*(A).

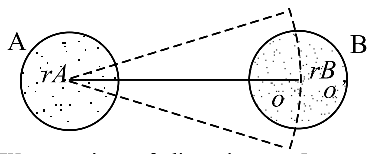


Figure 4. Illustration of *direction* and *reverse direction*

Following a similar intuition, the next notion we define is the notion of a border distance of a Data Bubble in the direction of another Bubble:

Definition 4.4 Let A and B be two sets of objects, represented by rA and rB, respectively. The *border distance of B in the direction of A* is defined as

$$Bubble(B).borderDistInDirection(A) := dist(rA, rB) - \min_{o \in B}(dist(o, rA))$$

The border distance of B in the direction of A is defined as the distance between the two representatives minus the distance between the representative of A, rA, and the object o in B that is closest to rA. Figure 5 illustrates our estimation of the border distance of a Data Bubble B in the direction of another Bubble A.

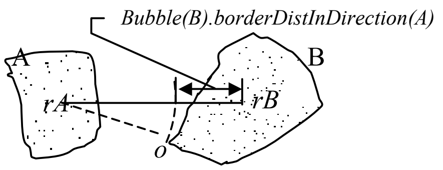


Figure 5. Illustration of *border distance*

A consequence of our definition of border distance is that – in contrast to what can happen in the original Data Bubbles – the extents of two Data Bubbles can never overlap, i.e., the distance from a representative rB of a Data Bubble B to its "border", in the direction of a Data Bubble A with representative rA, can never be larger than half the distance between the two representatives:

Lemma 4.1 Given two Data Bubbles A and B with representatives rA and rB, respectively. Let $Bubble(B).borderDistInDirection(A)$ be the border distance of B in the direction of A. If the distance function is a metric, i.e., satisfies the triangle inequality, then $Bubble(B).borderDistInDirection(A) \leq dist(rA, rB)/2$.

Proof. Suppose the border distance is greater than $dist(rA, rB)/2$. It follows that $dist(o, rA) < dist(rA, rB)/2$, where $o = \arg\min_{o \in B}(dist(o, rA))$. And because $o \in B$, by construction of B it must be $dist(o, rB) \leq dist(o, rA)$. But then it follows that $dist(o, rA) + dist(o, rB) \leq 2*dist(o, rA) < dist(rA, rB)$, which violates the assumption that the triangle inequality holds. Hence $Bubble(B).borderDistInDirection(A) \leq dist(rA, rB)/2$.

Our definition serves well in a "normal" situation of well-separated bubbles as depicted in Figure 6, where the distance between the "borders" of the bubbles gives a good estimate of the true distance between the point sets.

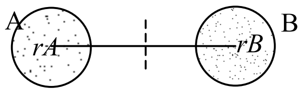


Figure 6. "Normal" Data Bubble separation

In practice, some situations can occurs where a Data Bubble may contain a "gap" in a particular direction. If the Data Bubbles represent points from different clusters but their representatives happen to be close enough so that one Data Bubble contains points from both clusters. Figure 7 illustrates such a case.

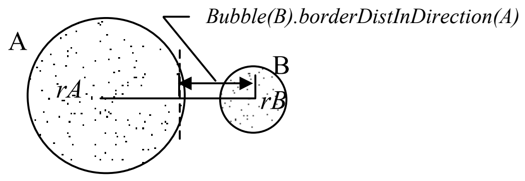


Figure 7. A"gap" in Data Bubble *B* (border distance in direction *A* is larger than in reverse direction of *A*)

These situations can lead to errors in the clustering result because the difference between the borders, and hence the distance between Data Bubbles may be underestimated. As a consequence, cluster separations may be lost. For vector Data Bubbles, this problem does not occurs as frequently as for non-vector Bubbles, since the extent is estimated by the average pair-wise distance, which in the case depicted in Figure 7 would still be close to the true extent of B (which is much smaller than the depicted directional border distance of B).

In order to detect those and similar situations, we maintain certain statistics with respect to the distances of objects in a Data Bubble B to its representative rB. The values we compute when constructing a Data Bubble are: 1) average distance of the objects in B in direction of each other Data Bubble (and reverse directions), 2) the standard deviation of the distances in all those directions.

Definition 4.5 Let A and B be two sets of objects, represented by rA and rB, respectively. Let B_A=*Bubble(B).InDirection(A)* denote the set of objects in B that lie in direction of A and let B_{revA}= *Bubble(B).InRevDirection(A)* denote the set of objects in B that lie in the reverse direction of A.

- $Bubble(B).aveDistInDirection(A) := \dfrac{\sum_{o \in B_A} dist(o, rB)}{|B_A|}$

Bubble(B).aveDistInDirection(A) is the average distance between the representative of B and the objects in B that lie in direction of A.

- $Bubble(B).aveDistInRevDirection(A) := \dfrac{\sum_{o \in B_{revA}} dist(o, rB)}{|B_{revA}|}$

Bubble(B).aveDistInRevDirection(A) is the average distance between the representative of B and the objects in B that lie in reverse direction of A.

Definition 4.6 Let A and B be two sets of objects, represented by rA and rB, respectively. Let again B_A=*Bubble(B).InDirection(A)* and B_{revA}= *Bubble(B).InRevDirection(A)*. Let furthermore,

$\overline{dist_{B_A}} = Bubble(B).aveDistInDirection(A)$ and

$\overline{dist_{B_{revA}}} = Bubble(B).aveDistInRevDirection(A)$

- $Bubble(B).stdevInDirection(A)$:=
  $$\sqrt{\frac{\sum_{o \in B_A} (dist(o,rB) - \overline{dist_{B_A}})^2}{|B_A|}}$$
  Bubble(B).stdevInDirection(A) is the standard deviation of the distances between the representative of B and the objects in B that lie in direction of A.
- $Bubble(B).stdevInRevDirection(A)$:=
  $$\sqrt{\frac{\sum_{o \in B_{revA}} (dist(o,rB) - \overline{dist_{B_{revA}}})^2}{|B_{revA}|}}$$
  Bubble(B).stdevInRevDirection(A) is the standard deviation of the distances between the representative of B and all objects in B lying in reverse direction of A.

When constructing a Data Bubble B, the defined averages and standard deviations of distances relative to other bubbles A can be maintained incrementally:

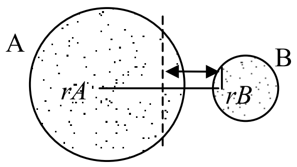
Lemma 4.2 Given a set of distance values $D = (d_1, d_2, \ldots, d_n)$. Let lsD denote the linear sum of the distance values in D, i.e., $lsD = \sum_{i=1\ldots n} d_i$, and let ssD denote the square sum of the distance values in D, i.e., $ssD = \sum_{i=1\ldots n}(d_i)^2$. Then, the average distance d_{ave} in D can be computed as $d_{ave}=lsD/n$, the standard deviation of the distances can be computed as $\sigma = \sqrt{\frac{n \cdot ssD - (lsD)^2}{n^2}}$.

Proof. By simple arithmetic transformations.

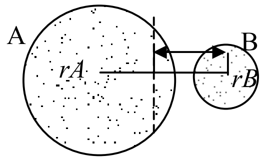
Since a linear sum and a square sum of distances in certain directions can be computed incrementally while constructing Data Bubbles, it follows that also the average and the standard deviation of those distances can be maintained incrementally without much computationally overhead: all needed distances are computed anyway in the classification step of the Data Bubble creation.

The "directional" versions of border distance, average distance and standard deviation of the distances help us to detect "gaps" in a Data Bubble in many cases. The situation depicted in Figure 7, e.g., is indicated by the fact that the border distance of Bubble B in direction of A is both much larger than in the reverse direction and much larger than the average distance in direction of A. Two other examples of "gaps" in a bubble are given in Figure 8.

In order to avoid overestimating the extent of a Data Bubble (and consequently underestimating the distance between Data Bubbles) in the presence of "gaps", we introduce a refined notion of "directional" border distance, which we call *"directional extent"* of the Data Bubble.



(a) Average distance of B in direction of A is larger than in the reverse direction



(b) Standard deviation of B in direction of A is much larger than in the reverse direction

Figure 8. Examples of a "gap" in a Data Bubble B

Definition 4.7 Let A and B be two sets of objects, represented by rA and rB, respectively. Let Ave, and $Stdv$ be the average respectively the standard deviation of the distances in B in direction of A or the reverse direction of A – whichever is smaller. The *extent of B in the direction of A*, *Bubble(B).extentInDirection(A)*, is then defined as

Bubble(B).extentInDirection(A):=
$$\min(Bubble(B).borderDistInDirection(A),\ Ave + 2 \cdot Stdv)$$

Basically, the (directional) extent of a Data Bubble is either the (directional) border distance, or the (directional) average distance plus two times (directional) standard deviation – whichever is smaller. Taking the average distance plus two times standard deviation is a way to estimate a ("directional") border around the representative that will include most of the points within that limit.

Having a good estimation of the extent of a Data Bubble in a certain direction, we can define the distance between two Data Bubbles simply as the distance between their representatives minus their directional extents.

Definition 4.8 Let A and B be two sets of objects, represented by rA and rB, respectively. The distance between A and B is defined as

$$\text{dist}(A,B)=\text{dist}(rA,rB)-Bubble(A).extentInDirection(B) -Bubble(B).extentInDirection(A)$$

In summary, our new method for constructing a collection of s Data Bubbles for non-vector data is as following:

1. Draw randomly a sample of s objects from the database. Each sample object will be the representative object rB for one of the s Data Bubbles B.
2. Classify, i.e., assign each object in the database to the closest representative object rB in the set of objects obtained in step 1, and maintain incrementally the following information about each Data Bubble B:
   a) The distances to the k-nearest neighbours of the representative object rB, up to a value k=*MinPts*. These k-nearest-neighbour distances, *nnDist*(k,B) are used to define core-dist and virtual reachability as in [3], i.e., *core-dist*(B)=*virtualReachability*(B)=*nnDist*(*MinPts*,B).

b) Relative to each other Data Bubble A:
  - $Bubble(B).borderDistInDirection(A)$
  - Average distance and standard deviation in direction of A and in reverse direction of A.

3. Compute the extent of each Data Bubble B in direction of every other Data Bubble A:

```
Ave:=min(Bubble(B).aveInDirection(A),
         Bubble(B).aveInRevDirection(A));
Dev:=min(Bubble(B).stdevInDirection(A),
         Bubble(B). stdevInRevDirection(A));
Bubble(B).extentInDirection(A):=
min(Bubble(B).borderDistInDirection(A), Ave + 2*Dev).
```

After the construction of the Data Bubbles and the computation of the directional extent values, hierarchical algorithms such as the Single-Link method can be applied to the non-vector Bubbles by using the distance between Data Bubbles defined in definition 4.7.

The clustering algorithm OPTICS is based on the notion of reachability distance. For point objects, the reachability distance of an object o_1 relative to an object o_2 was defined as the maximum of dist(o_1, o_2) and core-distance(o_2) (see [1] for details). For Data Bubbles, the notion of the reachability distance of a Data Bubble B_1 relative to a Data Bubble B_2 can be defined analogously:

Definition 4. 9 Let A and B be two Data Bubbles,. The reachability distance of B relative to A is defined as

$$reach\text{-}dist(B,A)= \max(\text{dist}(rA,rB), \text{core-dist}(A), \text{core-dist}(B)).$$

This definition is an improved version of the definition used in [3], which estimates the reachability distance of a hypothetical object o in B_1 in direction of B_2, relative to an object in B_2 in direction of B_1. Analogously to the definition of reachability distance for points, if the two bubbles are far apart, the reachability distance will be equal to the distance between the bubbles. If the bubbles are very close to each other, which is indicated by at least one of the core-distances being larger than the distance between the bubbles, the hypothetical object o can be considered as being located at the border of both Data Bubbles, and we estimate its core-distance by the larger of the two core-distances, which in turn is used as the estimation of the reachability distance. The definition given in [3] only considers the core-distance of the Bubble A when estimating the reachability distance of Bubble B relative to A. The old definition underestimates the reachability value for points at the border of B significantly if A is relatively dense in its "centre" and close to a less dense Bubble B (resulting in errors in the clustering structure). Furthermore, this definition allows a much easier integration of Data Bubbles into OPTICS than the original method.

5. Performance Evaluation

In this section, we perform an extensive experimental evaluation of our Data Bubbles. The results show that our new method is highly scalable, that it produces reliable results even for very small numbers of Data Bubbles, that it is significantly better than random sampling, and that it even has an improved accuracy compared to the original Data Bubbles when applied to vector data.

5.1 Data Sets and Experimental Setup

We evaluate the performance of our new method for non-vector Data Bubbles using the following data sets:

First, a synthetic 2-dimensional point data set, called *DS-Vector*, which is used to show that even for a Euclidean vector space the new version of Data Bubbles (which only uses the distance information and none of the vector space properties) outperforms the original Data Bubbles (for a vector space) proposed in [3]. The reachability plot obtained when clustering the whole data set using OPTICS is depicted in Figure 9 (left). The data set contains 50000 points distributed over 8 clusters and 4% background noise. The eight clusters have similar sizes and most of them are located very close to each other as can be seen from the relatively low reachability values that separate most of them. Therefore, this data set is a good test bed for evaluating the new directional definition of extent and the heuristics to handle gaps in bubbles.

The second data set, called *DS-Tuple*, which we use to evaluate the relative performance of our non-vector Data Bubbles, is a synthetic set of binary strings. Each object of DS-tuple is a 100-bit 0/1 sequence, and the similarity between two such sequences s_1 and s_2 is measured using the Jaccard coefficient, i.e. $|s_1 \cap s_2| / |s_1 \cup s_2|$. 80% of the objects form 10 clusters and the remaining 20% are noise. Two of the clusters are very small (123 and 218 objects), making the problem of finding them very challenging for data summarization techniques. The reachability plot obtained when clustering the whole data set using OPTICS is depicted in Figure 9 (right) (the two tiny clusters are indicated by arrows).

The third data set used to illustrate the practical relevance of our method is a real data set containing RNA sequences. The application and the data set, called *DS-Real*, are explained in more detail in section 5.2.

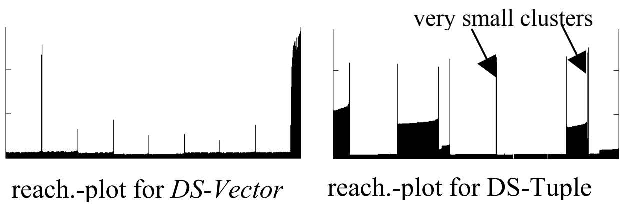


Figure 9. Reachability plots for the whole synthetic data sets used for the evaluation

The values reported in the following sections for the synthetic data sets are average values over 1000 repetitions of each experiment. In order to measure the quality of the hierarchical clustering results based on data summarizations, we use the following scoring scheme: For each reachability plot obtained for data summarizations, we apply a heuristic to select the best cut-line through the diagram. The heuristics evaluates 40 different cut-lines through a reachability plot in equidistant intervals. Each cut-line is assigned a score based on the number of clusters that are present with respect to this cut through the reachability plot. If k clusters are found ($0 \leq k \leq$ maximum number of clusters in the original data set, *k_max*), then the cut-line gets a score of *k/k_max*. If k is greater than *k_max*, it gets a score of 0. Hence missing clusters, finding spurious clusters, and splitting clusters gets penalities. The intention of this score is to penalize especially those results where the algorithm produces structures that are not existent in the original data set and which may lead to misinterpretations of a data set. The cut-line with the best score is selected as an approximation of the cut that a user would select as the best one.

All experiments were performed on an AMD Athlonxp1800+ workstation with 512 MB of memory.

5.2 Experimental Results

Comparison with original Data Bubbles

First, we compare our new method with the original Data Bubble method using the vector data set DS-Vector. The scores for both methods when increasing the number of Data Bubbles are depicted in Figure 10. The results clearly show that our new directional definition of extent and the heuristics to handle gaps in Data Bubbles leads to a better quality of the results, even when not using any of the vector space properties.

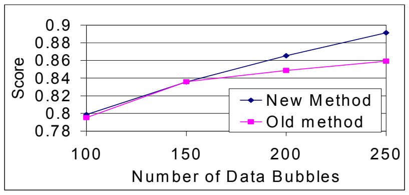


Figure 10. New versus old method using vector data

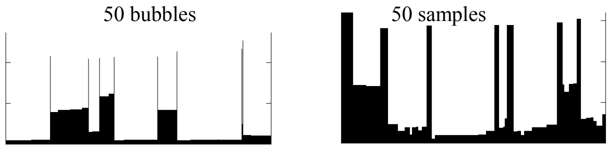
Comparison with random sampling

In this section we apply our method to the non-vector data set DS-Tuple. We evaluate both the quality and the runtime benefits of non-vector Data Bubbles. The first set of experiments shows the relative performance of our method compared to a random sampling approach, which just clusters a random sample of the data set and then assigns every object in the database to the closest sample object. This approach represents a baseline method, which is in fact difficult to beat since it is very efficient. If a method is computationally relatively expensive in the construction of its data summarizations (such as the Birch based methods BUBBLE and BUBBLE-FM), random sampling can be more effective since it can use the same amount of resources to simply cluster a larger sample.

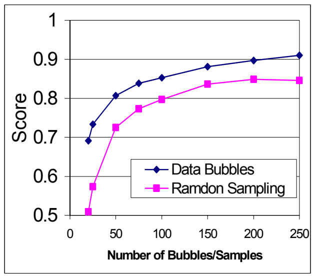
Figure 11 shows the result of the comparison on DS-tuple. The quality of plots created by our method is consistently better than that of random sampling. For example, when using 50 bubbles/samples, our method is almost always able to recover all the significant clusters and finds one of the 2 small ones regularly, while random sampling recovers only 7 of the 8 big clusters quite consistently. Two example plots are depicted in Figure 11 (a).

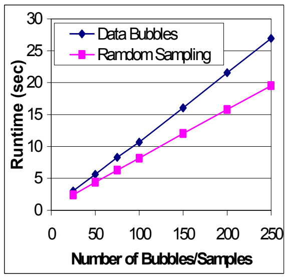
Figure 11 (b) shows the average score for both methods when varying the number of bubbles/samples. We obtain an up to 40% better quality when using very high compression rates (low numbers of bubbles). In general, we consistently obtain a score that can be obtained by random sampling only when the number of sample points is at least twice the number of Data Bubbles (this rate increases with the number of bubbles/samples).

Figure 11 (c) shows the scale-up of both methods with respect to the number of bubbles/samples. Both methods scale linearly. They both have the same amount of distance computation and hence their runtime is very close to each other, especially when the sampling rate is low. Random sampling is slightly faster when using a large sample rate and a relatively cheap distance function (Jaccard coefficient in this case). In real applications, however, the distance function is typically much more expensive (e.g., a sequence alignment score as in our real world data set DS-Real), and the runtime of both methods will be dominated heavily by only the distance computation (e.g., 638sec for Data Bubbles versus 635sec for sampling on the DS-Real data set).



(a) reachability-plots for bubbles and sampling

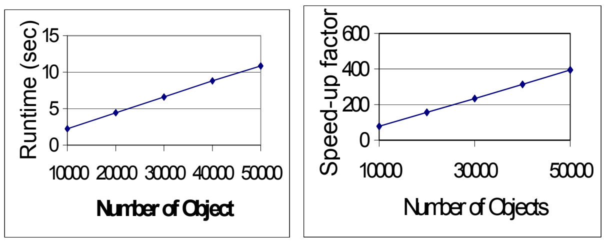




(b) quality (c) runtime

Figure 11. non-vector data bubbles vs. random sampling on the DS-Tuple data set.

Figure 12 shows the absolute runtime and the speed-up factors (compared to OPTICS on the whole database), when varying the database size, using subsets of DS-Tuple. Our algorithm (100 data bubbles) scales linearly with the size of database, and we achieve as expected large speed-up values over OPTICS: between 77 and 400. Note that this speed-up is also dependent on the distance function, and for much more expensive distance functions the expected speed-up will be much larger.



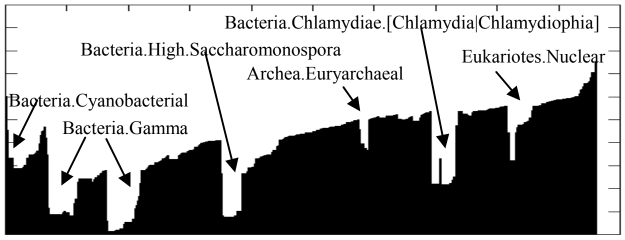
(a) scale-up (b) speed-up

Figure 12. Scale-up speed-up w.r.t number of objects

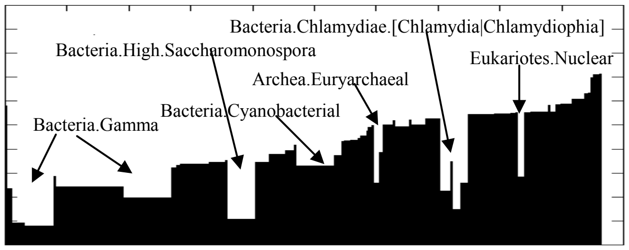
An Application to a Real Data Set

The RNase P Database [13] is a compilation of ribonuclease P (RNase P) sequences and other information. In the last a few years, the number and diversity of RNase P RNA sequences available have increased significantly and analysing this data set has become an important issue. Clusteranalysis can help detecting functional subgroups in this data set and help understanding the evolutionary relationships between the sequences.

In this application, we used global sequence alignment under the standard affine gap penalty scoring scheme (used in BLAST) to cluster the sequence database. The OPTICS result for the whole data set is shown in Figure 13(a). Figure 13(b) shows a good result using 50 Data Bubbles. It is easy to verify that the results are very similar. The clustering structure corresponds mostly to the already known evolutionary relationships and matches well with the annotations in the database. An exception is the *Bacteria.Gamma* family that is partitioned into two sub-groups, which both are mixed with respect to the existing annotations of the sequences. This is an interesting finding that is currently investigated in more detail.



(a) Result of OPTICS DS-Real, runtime = 6578 sec



(b) Result of OPTICS-Bubbles, runtime = 638 sec

Figure 13: Results for the RNA data set

6. Conclusions

In this paper, we presented a new data summarization method for non-vector data. The method uses only distance information, and introduces the novel concept of a directional extent of a set of objects. We show that the distance between bubbles based on this notion of extent even improves upon Data Bubbles when applied to vector data. An extensive performance evaluation also shows that our method is more effective than a random sampling approach, using only a very small number of data summarizations, and thus resulting in a large reduction of runtime (up to 400 times) while trading only very little clustering quality. The method allows us to obtain results even for data sets where clustering the whole data set is infeasible because of the prohibitive cost of the distance function.

Acknowledgements

- We would like to thank Yiping Zhou for fruitful discussions in the early stages of this project, and Dr. Guohui Lin for providing code and advice regarding the real data set and the sequence alignment.
- Research partially funded by NSERC Canada.

References

[1] Ankerst M., Breunig M. M., Kriegel H.-P., Sander J. OPTICS: Ordering Points To Identify the Clustering Structure. SIGMOD 1999, pp. 49-60.

[2] Bradley P. S., Fayyad U., Reina C.: Scaling Clustering Algorithms to Large Databases. KDD 1998, pp. 9-15.

[3] Breunig M., Kriegel H.-P., Kröger P., Sander J.: Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering. SIGMOD 2001, pp. 79-90.

[4] DuMouchel W., Volinsky C., Johnson T., Cortez C., Pregibon D.: Sqashing Flat Files Flatter. KDD1999, pp. 6-15.

[5] Faloutsos C., Lin K.-I.: Fastmap: A fast algorithm for indexing, datamining and visualization of traditional and multimedia databases. SIGMOD 1995, pp. 163-174.

[6] Ganti V., Ramakrishnan R., Gehrke J., Powell A., French J.: Clustering Large Datasets in Arbitrary Metric Spaces. ICDE 1999, pp. 502-511.

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

[8] Jin W., Tung A. K. H., Han J.: Mining top-n local outliers in large databases. KDD 2001, pp. 293-298.

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

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

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

[12] Zhang T., Ramakrishnan R., Linvy M.: BIRCH: An Efficient Data Clustering Method for Very Large Databases. SIGMOD 1996, pp. 103-114.

[13] RNase P Database, North Carolina State University, http://www.mbio.ncsu.edu/RNaseP/home.html