Published in Special Issue on Data Mining, KI-Journal, ScienTec Publishing, Vol. 1, 1998

Clustering for Mining in Large Spatial Databases

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu

In the past few decades, clustering has been widely used in areas such as pattern recognition, data analysis, and image processing. Recently, clustering has been recognized as a primary data mining method for knowledge discovery in spatial databases, i.e. databases managing 2D or 3D points, polygons etc. or points in some *d*-dimensional feature space. The well-known clustering algorithms, however, have some drawbacks when applied to large spatial databases. First, they assume that all objects to be clustered reside in main memory. Second, these methods are too inefficient when applied to large databases. To overcome these limitations, new algorithms have been developed which are surveyed in this paper. These algorithms make use of efficient query processing techniques provided by spatial database systems.

1. Introduction

Both, the number of databases and the amount of data stored in a single database, are rapidly growing [FPS 96]. Databases on sky objects, e.g., consist of billions of entries extracted from images generated by large telescopes. The NASA Earth Observing System, e.g., is projected to generate some 50 GB of remotely sensed data per hour. This growth of databases has far outpaced the human ability to interpret this data, creating a need for automated analysis of databases. *Knowledge Discovery in Databases* (*KDD*) [FPS 96] is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. A lot of research has been conducted on knowledge discovery in *relational* databases (*cf.* [CHY 96] for a survey). In this paper, we deal with knowledge discovery in *spatial* databases (*cf.* [KAH 96] for an overview), i.e. databases managing 2D or 3D points, polygons etc. or points in some *d*-dimensional feature space.

The KDD process is interactive and iterative, involving numerous steps [FPS 96]. *Focusing* is the step of selecting a subset of the data or of the attributes (variables), on which discovery is to be performed. *Data mining* is the step of applying data analysis algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data.

This paper focuses on clustering as one of the important data mining methods [FPS 96]. *Clustering* is the task of grouping the objects of a database into meaningful subclasses. Applications of clustering in spatial databases are, e.g., the detection of seismic faults by grouping the entries of an earthquake catalog [AF 96] or the creation of thematic maps in geographic information systems by clustering feature spaces [Ric 83].

The well-known clustering algorithms such as k-means [Mac 67], k-medoids [KR 90] and Single Link [Sib 73] have severe drawbacks when applied to large databases. First, they assume that all objects to be clustered can reside in main memory at the same time. Despite growing main memories, this assumption is not always true for large databases. Second, they are too inefficient on large databases. Therefore, new database-oriented clustering methods have been developed which are surveyed in this paper.

The rest of the paper is organized as follows. Section 2 briefly introduces spatial query processing. Section 3 shows how clustering algorithms can be integrated with spatial database systems. Section 4 discusses the clustering properties of spatial index structures and gives examples of clustering algorithms utilizing these properties. Section 5 concludes the paper and discusses several issues for future research.

2. Efficient Query Processing in Spatial Database Systems

Spatial database systems offer the underlying database technology for storing and retrieving spatial data efficiently. In this section, we sketch some important spatial query types and spatial index structures for efficient query processing.

Numerous applications, e.g. geographic information systems and CAD systems, require the management of spatial data such as points, lines and polygons. Note that the space of interest may either be an abstraction of a real *2D* or *3D* space such as a part of the surface of the earth or some *d*-dimensional space of feature vectors each describing an object of an application.

A *spatial database system* (*SDBS*) is a database system offering spatial datatypes in its data model and query language and supporting efficient implementation of these datatypes with their operations and queries [Gue 94]. The basic 2D datatypes, e.g., are points, lines and regions. Typical operations on these datatypes are the calculation of the distance or the intersection. Important query types are, e.g.:

- *region queries*, obtaining all objects intersecting a specified query region and
- *nearest neighbor* (*NN*) queries, obtaining the objects closest to a specified query object.

A trivial implementation of spatial queries would scan the whole database and check the query condition on each object. In order to speed up query processing, many spatial index structures have been developed to restrict the search to the relevant part of the space (*cf.* [Gue 94] for a survey). All index structures are based on the concept of a *page*, which is the unit of transfer between main and secondary memory. Typically, the number of page accesses is used as a cost measure for database algorithms because the runtime for a page access exceeds the runtime of a CPU operation by several orders of magnitude. Spatial index structures can be classified as organizing the data space (hashing) or organizing the data itself (search trees). In the following, we will introduce a well-known representative of each class. We choose the grid file and the R*-tree because they will be used in the following sections.

The *grid file* [NHS 84] has been designed to manage points in some *d*-dimensional data space generalizing the idea of 1-dimensional hashing. It partitions the data space into cells using an irregular grid. The split lines extend through the whole space and their positions are kept in a separate scale for each dimension. The *d* scales define a *d*-dimensional array, the directory, containing a pointer to a page in each cell. All *d*-dimensional points contained in a cells are stored in the respective page. In order to achieve a sufficient storage utilization of secondary memory, several neighboring cells (forming a rectilinear region) of the directory may be mapped to the same data page (see figure 1). Region queries can be answered by determining from the directory the set of grid cells intersecting the query region, by following the pointers to the corresponding data pages and then examining the points in these pages. Since the number of grid cells may grow exponentially with increasing dimension *d*, the grid file is efficient only for small values of *d*.

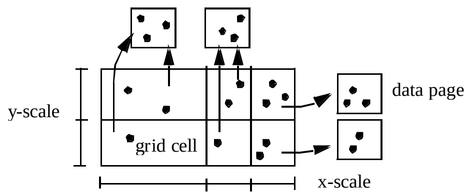

Figure 1: Sample grid file

The *R*-tree* [BKSS 90] generalizes the 1-dimensional B-tree to *d*-dimensional data spaces, specifically an R*-tree manages *d*-dimensional hyperrectangles instead of 1-dimensional numeric keys. An R*-tree may organize extended objects such as polygons using *minimum bounding rectangles* (*MBR*) as approximations as well as point objects as a special case of rectangles. The leaves store the MBR of data objects and a pointer to the exact geometry. Internal nodes store a sequence of pairs consisting of a rectangle and a pointer to a child node. These rectangles are the MBR's of all data or directory rectangles stored in the subtree having the referenced child node as its root (see figure 2). To answer a region query, starting from the root, the set of rectangles intersecting the query region is determined and then their referenced child nodes are searched until the data pages are reached. Since the overlap of the MBR's in the directory nodes grows with increasing dimension *d*, the R*-tree is efficient only for moderate values of *d*. Following the ideas of R-trees, recently index structures have been designed which are also efficient for larger values of *d* [BKK 96].

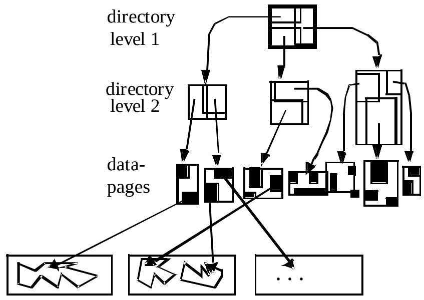

Figure 2: Sample R*-tree

3. Clustering Algorithms Integrated with SDBS

There are two well-known types of clustering algorithms: partitioning and hierarchical algorithms [KR 90]. Recently, single scan clustering algorithms have been introduced yielding a significant gain in efficiency compared to the other types of algorithms. In this section, we present partitioning, hierarchical and single scan algorithms which are integrated with *SDBS* by spatial index structures for efficient data mining.

3.1 Partitioning Clustering Algorithms

Partitioning algorithms construct a partition of a database *D* of *n* objects into a set of *k* clusters where *k* is an input parameter. Each cluster is represented by the gravity center of the cluster (*k*-means) [Mac 67] or by one of the objects of the cluster located near its center (*k*-medoid) [KR 90] and each object is assigned to the cluster with the closest representative. This implies that the resulting clustering is equivalent to a Voronoi diagram of all representatives. Thus, the shape of all clusters found by a partitioning algorithm is convex (see figure 3).

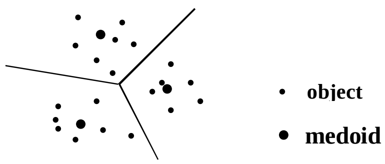

Figure 3: A clustering with n=23 objects and k=3 clusters

Partitioning algorithms typically start with an initial partition of *D* and then use an iterative control strategy to optimize the clustering quality, e.g. the average distance of an object to its representative. Thus, partitioning algorithms consider clustering as an optimization problem and they may suffer from the problem of local minima. The first use of a partitioning clustering algorithm for spatial data mining is presented in [NH 94]. The algorithm *CLARANS* (*Clustering Large Applications based on RANdomized Search*) is introduced, which is a *k*-medoid algorithm with a randomized and bounded search strategy for improving the initial partition. CLARANS is significantly more efficient than the well-known *k*-medoid algorithms PAM and CLARA presented in [KR 90], but it is still too inefficient to be applied to large databases.

In [EKX 95] two techniques to integrate CLARANS with an SDBS using a spatial index structure are proposed. The first is *R*-tree* based sampling (see section 4.1), the second is *focusing on relevant clusters* which is described in the following.

Typically, *k*-medoid algorithms try to improve a current clustering by exchanging one of the medoids of the partition with one non-medoid and then compare the quality of this "new" clustering with the quality of the "old" one. In CLARANS, computing the quality is the most time consuming step because a scan through the whole database is performed. However, only objects which belonged to the cluster of the exchanged medoid or which will belong to the cluster of the new medoid contribute to the change of quality. Thus, only the objects of 2 (out of *k*) clusters have to be read from disk. Assuming the same average size for all clusters, a performance gain of *k/2* (measured by the number of page accesses) compared to [NH 94] is expected. To retrieve exactly the objects of a given cluster, a region query can be used. This region, a Voronoi cell whose center is the medoid of the cluster, can be efficiently constructed from the medoids and the MBR of all objects in the database.

Partitioning algorithms are well-suited if the clusters are of convex shape and similar size and if their number *k* can be reasonably estimated. If *k* is not known, various parameter values can be tried and for each of the discovered clusterings the *silhouette coefficient* [KR 90] - a numerical measure of the appropriateness of the number of clusters - can be calculated.

3.2 Hierarchical Clustering Algorithms

Whereas partitioning algorithms obtain a single level clustering, *hierarchical algorithms* decompose a database *D* of *n* objects into several levels of partitionings (clusterings). The hierarchical decomposition is represented by a *dendrogram*, a tree that iteratively splits *D* into smaller subsets until each subset consists of only one object. In such a hierarchy, each node of the tree represents a cluster of *D*. Figure 4 illustrates the hierarchical clustering process. The dendrogram can either be created from the leaves up to the root (*agglomerative approach*) or from the root down to the leaves (*divisive approach*) by merging resp. dividing clusters at each step.

In contrast to partitioning algorithms, hierarchical algorithms do not need *k* as an input. However, a *termination condition* has to be defined indicating when the merge or division process should be terminated. One example of a break condition in the agglomerative approach is the critical distance D_{min} between all the clusters of *Q*. If no distance between two clusters of *Q* is smaller than D_{min}, then the algorithm terminates. Alternatively, an appropriate level in the dendrogram has to be selected manually after the creation of the whole dendrogram.

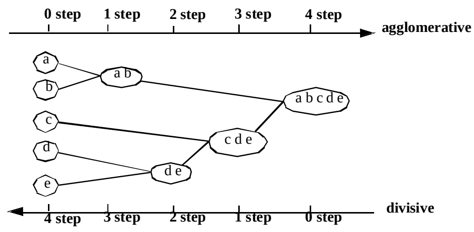

Figure 4: Hierarchical clustering of the set {a, b, c, d, e}

Agglomerative hierarchical clustering algorithms keep merging the closest pairs of objects to form clusters. These algorithms are based on the inter-object distances and on finding the nearest neighbors (*NN*) of objects. The time 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 *NN*.

Murtagh [Mur 83] points out that spatial index structures make use of the fact that finding *NN*'s is a "local" operation since the *NN* of an object can only lie in some restricted region of the data space. Thus, using multi-dimensional hash- or tree-based index structures for efficient processing of *NN* queries can significantly improve the overall run-time complexity of agglomerative hierarchical clustering algorithms. If a disk-based index structure, e.g. a grid file or R*-Tree, is used instead of a main-memory-based index structure, these hierarchical clustering algorithms can also be integrated with *SDBSs*.

3.3 Single Scan Clustering Algorithms

The basic idea of a *single scan algorithm* is to group neighboring objects of the database into clusters based on a local cluster condition thus performing only one scan through the database. The result is, like in the case of partitioning algorithms, a single level clustering. Single scan clustering algorithms are very efficient if the retrieval of the neighborhood of an object is efficiently supported by the SDBS, i.e. if the average runtime complexity of a region query is $O(\log n)$ for a database of *n* objects. Then, the overall runtime complexity of a single scan algorithm is $O(n \log n)$. Furthermore, if the runtime complexity for retrieval of a neighborhood is $O(1)$, e.g. for low dimensional raster or grid data, then the overall runtime complexity of a single scan algorithm is only $O(n)$. On the other hand, if the runtime complexity for retrieval of the neighborhood is $O(n)$, e.g. in a very high-dimensional feature space where the performance of spatial index structures degrades, then the runtime complexity of a single scan algorithm becomes $O(n^2)$.

The algorithmic schema of a single scan clustering algorithm is as follows:

```
SingleScanClustering(Database DB)
    FOR each object o in DB DO
        IF o is not yet member of some cluster THEN
            create a new cluster C;
            WHILE neighboring objects satisfy the
                    cluster condition DO
                add them to C
            ENDWHILE
        ENDIF
    ENDFOR
```

Different cluster conditions yield different cluster definitions and algorithms. In the following, we present two single scan algorithms.

DBSCAN (Density Based Spatial Clustering of Applications with Noise) [EKSX 96] relies on a density-based notion of clusters and is designed to discover clusters of arbitrary shape in spatial databases with noise. The key idea is that for each point of a cluster the neighborhood of a given radius (*Eps*) has to contain at least a minimum number of points (*MinPts*), i.e. the density in the neighborhood has to exceed some threshold. DBSCAN needs two parameters. In [EKSX 96], a simple heuristic is presented which is effective in many cases to determine the parameters *Eps* and *MinPts* of the "thinnest" cluster in the database. If the user can give an estimation of the percentage of the noise, the heuristic helps the user to manually choose *Eps* and *MinPts* through the visualization of the distance to the k-th nearest neighbor for the points in the database.

The cluster condition of DBSCAN can be generalized in the following ways [EKSX 97]: First, any notion of a neighborhood of an object can be used if it is based on a binary predicate which is symmetric and reflexive. Second, instead of simply counting the objects in a neighborhood of an other measures to define the “cardinality” of that neighborhood can be used. A distance-based neighborhood is a natural notion of a neighborhood for point objects, but it is not clear how to apply it for the clustering of spatially extended objects such as a set of polygons of considerably differing sizes. Neighborhood predicates such as *intersects* or *meets* may be more appropriate for finding clusters of polygons in many cases. When clustering objects represented by polygons, the area of the polygons may, e.g., be used as a weight for the objects in the definition of the “cardinality”.

In [XEKS 98] the non-parametric clustering algorithm *DBCLASD (Distribution Based Clustering of LArge Spatial Databases)* is proposed. For many applications, the assumption is quite reasonable that the points inside of a cluster are randomly distributed ([AF 96], [BR 96]). This implies a characteristic probability distribution of the distance to the nearest neighbors for the points of a cluster. DBCLASD incrementally augments an initial cluster by its neighboring points as long as the nearest neighbor distance set of the resulting cluster still fits the expected distribution. DBCLASD detects clusters of arbitrary shape without requiring any input parameters if the points inside of the clusters are almost randomly distributed.

The algorithms CLARANS, DBSCAN and DBCLASD were evaluated both on synthetic and real 2D databases ([EKSX 96], [XEKS 98]). The evaluation is based on implementations in C++ using the R*-tree and the experiments were run on HP C160 workstations. Table 1 summarizes the results for several test databases. The run time of DBSCAN is only slightly higher than linear in the number of points and the runtime of DBCLASD is between 1.5 and 3 times the runtime of DBSCAN. This factor increases with increasing size of the database. The run time of CLARANS is close to quadratic in the number of points.

Table 1: Comparison of the runtime (in sec.)

No. of points	CLARANS	DBCLASD	DBSCAN
5000	3019	46	31
10000	13600	95	50
15000	38828	137	80
20000	83133	209	110
25000	150474	274	134
100000	not measured	1633	1130
200000	not measured	3974	1654
300000	not measured	6137	2446
400000	not measured	9048	3334
500000	not measured	12943	4184

4. Exploiting the Clustering Properties of Index Structures

In this section, we show how spatial index structures and similar data structures can be used for clustering very large databases. These structures organize the data points or the data space in a way that points which are close to each other are grouped together on a disk page. Thus, index structures contain much information about the distribution of the points and their clustering. These structures can be used for special clustering algorithms which take advantage of the information stored in the directory of the index or they can be used as a kind of preprocessing for other clustering algorithms.

4.1 R*-Tree-Based Sampling

To cluster *SDBS* in limited main memory, one can select a relatively small number of representatives from the database and apply the clustering algorithm only to these representatives. This is a kind of data sampling, a technique common in cluster analysis [KR 90]. The drawback is that the quality of the clustering will decrease when considering only a subset of the database.

Traditional data sampling works only in main memory. *Ester et al*. propose a new method of selecting representatives from a SDBS [EKX 95]. From each data page of an R*-tree, one or several representatives are selected. The clustering strategy of the R*-tree, which minimizes the overlap between directory rectangles, yields a well-distributed set of representatives (see figure 5). This is confirmed by their experimental results which show that the efficiency is improved by a factor of 48 to 158 whereas the clustering quality decreases only by 1.5 % to 3.2 % when comparing the clustering algorithm CLARANS with and without R*-tree-based sampling.

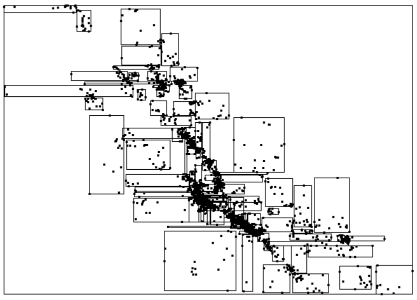

Figure 5: Data page structure of an R*-tree

4.2 Grid clustering

Schikuta [Sch 96] proposes a hierarchical clustering algorithm based on the grid file. Points are clustered according to their pages in the grid structure. The algorithm consists of 4 steps:

- Creation of the grid structure
- Sorting of the grid data pages according to page densities
- Identifying cluster centers
- Recursive traversal and merging of neighboring pages

In the first part, a grid structure is created from all points which completely partitions the data space into a set of non-empty disjoint rectilinear shaped data pages containing the points. Because the grid structure adapts to the distribution of the points in the data space, the creation of the grid structure can be seen as a pre-clustering phase (see figure 6).

In the second part, the grid data pages (containing the points from one or more grid cells) are sorted according to their density, i.e. the ratio of the actual number of points contained in the data page and the spatial volume of the page. This sorting is needed for the identification of cluster centers in the third part.

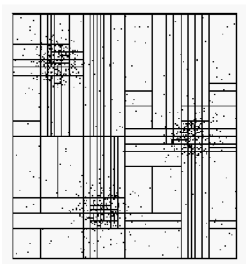

Figure 6: Data page structure of a grid file [Sch96]

Part 3 selects the pages with the highest density as cluster centers (obviously a number of pages will have the same page density). Step 4 is performed repeatedly until all cells have been clustered. Starting with cluster centers, neighboring pages are visited and merged with the current cluster if they have a lower or equal density than the current page. Then the neighboring pages of the merged neighbors are visited recursively until no more merging can be done for the current cluster. Then the next unclustered page with highest density is selected. Experiments [Sch 96] show that this clustering algorithm clearly outperforms hierarchical and partitioning methods of the commercial statistical package SPSS in terms of efficiency.

4.3 CF-Tree

Zhang *et al*. [ZRL 96] present the clustering method *BIRCH* (*Balanced Iterative Reducing and Clustering using Hierarchies*) which uses a highly specialized tree-structure for the purpose of clustering very large sets of points. The advantage of this structure is that its memory requirements can be adjusted to the main memory that is available.

BIRCH incrementally computes compact descriptions of subclusters, called Clustering Features $CF = (n, \Sigma x_i, \Sigma x_i^2)$ that contain the number of points, the linear sum and the square sum of all points in the cluster. The *CF*-values are sufficient for computing information about subclusters like centroid, radius and diameter and constitute an efficient storage method since they summarize information about subclusters instead of storing all points.

The *CF*-values are organized in a balanced tree with branching factor B and a threshold T (see figure 7). A nonleaf node represents a cluster consisting of all 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. Thus, the parameter T determines the size of the tree.

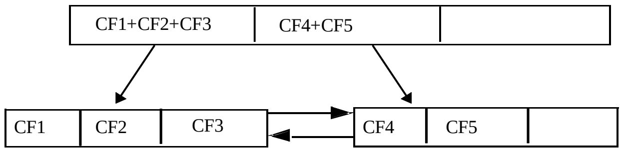

Figure 7: CF-tree structure

In the first phase, BIRCH performs a linear scan of all data points and builds a *CF*-tree. A point is inserted by searching the closest leaf of the tree. If an entry in the leaf can absorb the new point without violating the threshold condition, then the *CF*-values for this entry are updated, otherwise a new entry in the leaf node is created. In this case, if the leaf node contains more than L entries after insertion, the leaf node and possibly its ancestor nodes are split. In an optional phase 2 the *CF*-tree can be further reduced until a desired number of leaf nodes is reached. In phase 3 an arbitrary clustering algorithm (e.g. CLARANS) is used to cluster the *CF*-values of the leaf nodes.

Experiments with synthetic data sets show [ZRL 96] that the efficiency gain of BIRCH is similar to the gain of R*-tree-based sampling (see section 4.1) while the quality of the clustering using BIRCH in combination with CLARANS is even higher than the quality obtained by using CLARANS alone.

4.4 STING

Wang *et al*. [WYM 97] propose the *STING* (*STatistical INformation Grid based*) method which relies on a hierarchical division of the data space into rectangular cells. Each cell at a higher level is partitioned into a fixed number *k* of cells at the next lower level. The skeleton of the STING structure is similar to a spatial index structure - in fact, their default value for *k* is 4, in which case we have an equivalence for 2D data to the well known Quadtree structure [Sam 90]. This tree structure is further enhanced with additional statistical information in each cell of the tree (see figure 8). For each cell the following values are calculated and stored:

- *n* - the number of objects (points) in the cell.

  And for each numerical attribute:
- *m* - the mean of all values in the cell
- *s* - the standard deviation of all values in the cell
- *min* - the minimum value in the cell
- *max* - the maximum value in the cell
- *distr* - the type of distribution that the attribute values in this cell follow (enumeration type)

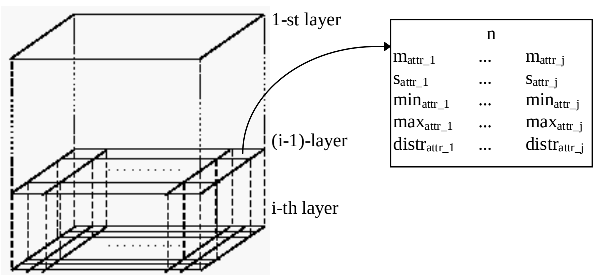

Figure 8: STING structure [WYM97]

The STING structure can be used to answer efficiently different kinds of region-oriented queries, e.g., finding maximal connected regions, which satisfy a density condition and possibly additional conditions on the non-spatial attributes of the points. The algorithm for answering such queries first determines all bottom level cells relevant to the query and then constructs regions of those relevant cells.

The bottom level cells that are relevant to a query are determined in a top down manner, starting with an initial layer in the STING structure - typically the root of the tree. The relevant cells in a specific level are determined using the statistical information. Then, the algorithm goes down the hierarchy by one level, considering only the children of relevant cells at the higher level. This procedure is iterated until the leaf cells are reached.

The regions of relevant leaf cells are then constructed by a breadth first search. For each relevant cell, cells within a certain distance are examined and merged with the current cell if the average density within the area is greater than a specified threshold. This is in principle the DBSCAN algorithm performed on cells instead of points. Wang et al. prove that the regions returned by STING are approximations of the clusters discovered by DBSCAN which become identical as the granularity of the grid approaches zero.

Wang et al. [WYM 97] claim that the runtime complexity of STING is O(b), where b is the number of bottom level cells. b is assumed to be much smaller than the number n of all objects which is reasonable for low dimensional data. However, to assure b << n for high dimensions *d*, the space cannot be divided along all dimensions: even if the cells are divided only once in each dimension, then the second layer in the STING structure would contain already 2^d cells. But if the space is not divided often enough along all dimensions, both the quality of cell-approximations of clusters as well as the runtime for finding them will deteriorate.

5. Conclusions

Recently, clustering has been recognized as a primary data mining method for knowledge discovery in spatial databases. The well-known clustering algorithms, however, have some drawbacks when applied to large spatial databases. First, they assume that all objects to be clustered reside in main memory. Second, these algorithms are too inefficient on large databases. To overcome these limitations, new techniques have been developed which were surveyed in this paper. We showed how to integrate clustering algorithms into a database management system and discussed how to exploit the clustering properties of spatial index structures.

There are several interesting issues for future research. In high-dimensional data spaces with a very large number of objects it becomes practically impossible to manually determine appropriate values for the input parameters of a clustering algorithm. Then, non-parametric algorithms are clearly advantageous.

So far, single scan algorithms create a one level clustering. However, a hierarchical clustering is more useful, in particular if the appropriate input parameters cannot be estimated accurately. It is possible to extend single scan algorithms like DBSCAN such that a hierarchy of clusterings is detected without much loss of efficiency.

Clustering may also be used for mining in data warehouses. Update operations are collected and applied to the data warehouse periodically and then all patterns (e.g. clusters) derived from the warehouse by some data mining algorithm have to be updated as well. Due to the very large size of many data warehouses, incremental clustering algorithms are highly desirable. The local nature of the cluster definition for single scan algorithms allows efficient incremental updates of a clustering.

References

[AF 96] Allard D. and Fraley C.:*"Non Parametric Maxi-*

mum Likelihood Estimation of Features in Spatial Point Process Using Voronoi Tessellation", Journal of the American Stat. Assoc., to appear in Dec. 1997. [http://www.stat.washington.edu/tech.reports/tr293R.ps].

[BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *"The R*-tree: An Efficient and Robust Access Method for Points and Rectangles"*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, ACM Press, 1990, pp. 322-331.

[BKK 96] Berchtold S., Keim D. A., Kriegel H.-P.: *"The X-Tree: An Index Structure for High-Dimensional Data"*, Proc. 22th Int. Conf. on Very Large Data Bases, Bombay, India, Morgan Kaufmann, 1996, pp. 28-39.

[BR 96] Byers S. and Raftery A. E.: "*Nearest Neighbor Clutter Removal for Estimating Features in Spatial Point Processes*", Tech. Report No. 305, Dep. of Statistics, Univ. of Washington. [http://www.stat.washington.edu/tech.reports/tr295.ps].

[CHY 96] Chen M.-S., Han J. and Yu P. S.: *"Data Mining: An Overview from Database Perspective"*, IEEE Trans. on Knowledge and Data Engineering, Vol. 8, No. 6, December 1996, IEEE Computer Society Press, pp 866-883.

[EKSX 97] Ester M., Kriegel H.-P., Sander J., Xu X.: *"Density-Connected Sets and their Application for Trend Detection in Spatial Databases"*. Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining. Newport Beach, Ca., AAAI Press, 1997.

[EKSX 96] 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, Oregon, AAAI Press, 1996.

[EKX 95] Ester M., Kriegel H.-P., Xu X.: "*Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification*", Proc. 4th Int. Symp. on Large Spatial Databases, Portland, ME, 1995, in: Lecture Notes in Computer Science, Vol. 951, Springer, 1995, pp.67-82.

[FPS 96] Fayyad, U. M., .J., Piatetsky-Shapiro, G., Smyth, P.: *"From Data Mining to Knowledge Discovery: An Overview"*, in: Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996 pp. 1-34.

[Gue 94] Gueting R. H.: *"An Introduction to Spatial Database Systems"*, in: The VLDB Journal, Vol. 3, No. 4, October 1994, pp.357-399.

[KAH 96] Koperski K., Adhikary J. and Han J.: "*Spatial Data Mining: Progress and Challenges"*, SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery, Montreal, Canada, 1996, Tech. Report 96-08, Univ. of British Columbia, Vancouver, Canada.

[KR 90] Kaufman L., Rousseeuw P. J.: *"Finding Groups in Data: An Introduction to Cluster Analysis"*, John Wiley & Sons, 1990.

[Mac 67] MacQueen, J.: *"Some Methods for Classification and Analysis of Multivariate Observations"*, 5th Berkeley Symp. Math. Statist. Prob., Volume 1, pp.281-297.

[Mur 83] Murtagh F.: *"A Survey of Recent Advances in Hierarchical Clustering Algorithms"*, The Computer Journal 26(4), 1983, pp.354-359.

[NH 94] Ng R. T., Han J.: *"Efficient and Effective Clustering Methods for Spatial Data Mining"*, Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile, Morgan Kaufmann, 1994, pp. 144-155.

[NHS 84] Nievergelt, J., Hinterberger, H., and Sevcik, K. C.: *"The Grid file: An Adaptable, Symmetric Multikey File Structure"*, ACM Trans. Database Systems 9(1), pp.38-71.

[Ric 83] Richards A.J.: "*Remote Sensing Digital Image Analysis. An Introduction*". Springer 1983.

[Sam 90] Samet H.: *"The Design and Analysis of Spatial Data Structures"*, Addison, 1990.

[Sch 96] Schikuta, E.: *"Grid clustering: An efficient hierarchical clustering method for very large data sets"*, In Proc. 13th Int. Conf. on Pattern Recognition, Volume 2, IEEE Computer Society Press, pp. 101-105.

[Sib 73] Sibson R.: "*SLINK: an optimally efficient algorithm for the single-link cluster method*". The Computer Journal 16(1), 1973, pp.30-34.

[WYM 97] Wang W., Yang J., Muntz R.: *"STING: A Statistical Information Grid Approach to Spatial Data Mining"*, Proc. 23rd Int. Conf. on Very Large Data Bases, Athens, Greece, Morgan Kaufmann, 1997, pp. 186-195.

[XEKS 98] Xu, X., Ester, M., Kriegel, H.-P., and Sander J.: *"A Nonparametric Clustering Algorithm for Knowledge Discovery in Large Spatial Databases"*, will appear in Proc. IEEE Int. Conf. on Data Engineering, Orlando, Florida, 1998, IEEE Computer Society Press.

[ZRL 96] 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, ACM Press, 1996, pp.103-114.