Proc. of the Fourth Int. Symposium on Large Spatial Databases (SSD '95),
Portland, Maine, USA, Lecture Notes in Computer Science, Springer, 1995.

Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification[1]

Martin Ester, Hans-Peter Kriegel, Xiaowei Xu

Institute for Computer Science, University of Munich
Leopoldstr. 11 B, D-80802 München, Germany
{ester | kriegel | xu}@informatik.uni-muenchen.de

Abstract. Both, the number and the size of spatial databases are rapidly growing because of the large amount of data obtained from satellite images, X-ray crystallography or other scientific equipment. Therefore, automated knowledge discovery becomes more and more important in spatial databases. So far, most of the methods for knowledge discovery in databases (KDD) have been based on relational database systems. In this paper, we address the task of class identification in spatial databases using clustering techniques. We put special emphasis on the integration of the discovery methods with the DB interface, which is crucial for the efficiency of KDD on large databases. The key to this integration is the use of a well-known spatial access method, the R*-tree. The focusing component of a KDD system determines which parts of the database are relevant for the knowledge discovery task. We present several strategies for focusing: selecting representatives from a spatial database, focusing on the relevant clusters and retrieving all objects of a given cluster. We have applied the proposed techniques to real data from a large protein database used for predicting protein-protein docking. A performance evaluation on this database indicates that clustering on large spatial databases can be performed, both, efficiently and effectively.

1 Introduction

Numerous applications require the management of geometric, geographic or *spatial* data, i.e. data related to space. The specific space may be, e. g. a two-dimensional projection of the surface of the earth, in a geographic information system, or a 3D space containing a protein molecule in an application in molecular biology. *Spatial Database Systems (SDBS)* [Gue 94] are database systems for the management of spatial data.

Both, the number and the size of spatial databases are rapidly growing because of the large amount of data obtained from satellite images, X-ray crystallography or other scientific equipment. This growth by far exceeds human capacities to analyze the databases in order to find implicit regularities, rules or clusters hidden in the data. Therefore, automated knowledge discovery becomes more and more important in spatial databases. *Knowledge discovery in databases (KDD)* is the non-trivial extraction of implicit, previously unknown, and potentially useful information from databases [FPM 91]. So far, most of the KDD methods have been based on relational database systems which are appropriate to handle non-spatial data, but not spatial data.

---

1. This research was funded by the German Minister for Research and Technology (BMFT) under grant no. 01 IB 307 B. The authors are responsible for the contents of this paper.

One of the well-known techniques for KDD is induction. [HCC 93] assumes the existence of concept hierarchies in the application domain and uses them to generalize the tuples of a relation into characteristic rules and classification rules. [LHO 93] extends this method for SDBS by adding spatial concept hierarchies and performing spatial induction. However, these hierarchies may not be available in many applications and, if available, they will not be appropriate for all KDD tasks. Therefore, [NH 94] does not rely on any domain knowledge and explores the applicability of cluster analysis techniques for KDD in SDBS. An algorithm called CLARANS (Clustering Large Applications based on RANdomized Search) is presented, which is both, efficient and effective for databases of some thousand objects.

[NH 94] assumes that all objects to be clustered can reside in main memory at the same time. However, this does not hold for large databases. Furthermore, the runtime of CLARANS is prohibitive on large databases. In general, the issue of interfacing KDD systems with a database management system (DBMS) has received little attention in the KDD literature and many systems are not yet integrated with a DBMS (c.f. [MCP 93]). [MCP 93] proposes an architecture of a KDD system including a DBMS interface and a focusing component. Well-known techniques are, e.g. focusing on a small subset of all tuples or focusing on a subset of all attributes. [AIS 93] presents a set of basic operations for solving different KDD tasks and shows how to apply them for efficient *classification*, i.e. finding rules that partition the database into a given set of groups. Good performance even on a large database is obtained by splitting the search space into independent parts, which is possible because different branches of a decision tree may be expanded independently from each other. [HK 94] addresses the issue of classification on large databases for relational database systems. Splitting the given relation into a lot of relatively small binary relations, i.e. focusing on one attribute at a time, [HK 94] always keeps the relevant part of the database in main memory. Additional histograms for each of the binary relations efficiently support the expensive computation of rule quality.

The task considered in this paper is *class identification*, i.e. the grouping of the objects of the database into meaningful subclasses (c.f. [MCP 93]).We show how to integrate CLARANS with a SDBS in order to perform class identification on large spatial databases, which can only partially be loaded into main memory. The key to this integration is the use of a well-known spatial access method, the R*-tree [BKSS 90]. The R*-tree, designed for supporting spatial queries, provides an efficient interface to a SDBS. This DB interface supports several focusing techniques allowing efficient class identification even on large spatial databases. The rest of the paper is organized as follows. Chapter 2 gives a brief introduction into CLARANS and discusses its application to large databases. An architecture for KDD in SDBS is outlined in chapter 3, giving special attention to the SDB interface. The focusing component of our KDD system, a main contribution of this paper, is presented in chapter 4. We perform an evaluation of both, efficiency and effectiveness on a protein database (chapter 5) and finish with the conclusion in chapter 6.

2 CLARANS on Large Databases

Cluster analysis techniques are attractive for KDD, because they can find hidden structures in data without using any additional domain knowledge. Different kinds of clustering algorithms have been developed (see [KR 90] for a survey), *k-mean* being one of the most prominent ones. The methods of type k-mean (e.g. ISODATA), however, suffer from some important drawbacks when applied to databases. First, these algorithm are quite sensitive to outliers, i.e. objects which are far away from the rest of the objects. Second, the cluster centers are no objects of the database, i.e. they may have no meaning in the domain of the application. Third, k-mean can only be applied when the mean of a cluster of objects is defined, which may not be the case in some applications, e.g. when the attributes are non-numeric. *k-medoid* algorithms avoid these drawbacks. Their goal is to find representative objects, called *medoids.* To achieve this goal, only the definition of a distance for any two objects is needed. Note that a distance function is also required by k-mean algorithms. Each object is assigned to the closest medoid so that the database of objects is partitioned into a set of clusters. Because of the above advantages, we have chosen the type k-medoid as the basis for our approach.

In the following, we introduce some basic notions for this paper. Let O be a *set of n objects*. $M \subseteq O$ denotes the set of *k* medoids*,* NM = O - M denotes the *set of non-medoids*. We assume the objects to be polyhedrons, a common assumption for SDBS. Thus, each object is given by a list of its edges, and an edge is given by a list of two vertices, being points. Let $P \subseteq R^3$ be the set of all *points*. In general, the objects are spatial, and we define the *center of an object* to be the arithmetic mean of its vertices.

$$\textit{center:}\ O \rightarrow P$$

Let *dist* be a *distance function*:

$$\textit{dist:}\ P \times P \rightarrow R_0^+$$

In the following, we assume *dist* to be the euclidean distance, which is a natural choice for spatial clustering. Distance function *dist* can naturally be extended from points to polyhedron objects via the center function:

$$\textit{dist:}\ O \times O \rightarrow R_0^+$$

$$\textit{dist}\ (o_i, o_j) = \textit{dist}\ (\textit{center}(o_i), \textit{center}(o_j))$$

Now each object is assigned to one of the medoids, such that the distance of its center to its medoid is minimal. Therefore, we define a function medoid:

$$\textit{medoid:}\ O \rightarrow M$$

$$\textit{medoid}\ (o) = m_i,\ m_i \in M,\ \forall\ m_j \in M\text{: } \textit{dist}\ (o, m_i) \leq \textit{dist}\ (o, m_j)$$

Finally, we define the *cluster* of medoid m_i to be the subset of all objects from O with medoid(o) = m_i and a *clustering* to be a set of clusters partitioning O. Let C_O be the set of all possible clusterings of O. The *total distance* of a clustering is used to measure its quality:

$$\textit{total\_distance:}\ C_O \rightarrow R_0^+$$

$$\textit{total\_distance}(c) = \sum_{m_i \in M} \sum_{o \in \textit{cluster}(m_i)} \textit{dist}\ (o, m_i)$$

PAM (Partitioning Around Medoids, see [KR 90]) is an algorithm of type k-medoid. It starts from an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids as long as the total distance of the resulting clustering is improved. PAM works efficiently for small data sets, but its runtime is prohibitive for large databases. [NH 94] proposes a clustering method called CLARANS (Clustering Large Applications based on RANdomized Search) based on PAM with a new heuristic search strategy. The key idea is not to consider all possible replacements of one medoid by one non-medoid and to select the optimal one, but to perform the first replacement improving the quality of the clustering. The clustering obtained after performing a single replacement is called a *neighbor* of the current clustering. The number of neighbors tried is restricted by a parameter provided by the user (*maxneighbor*) and the selection of these neighbors is random. Each iteration of CLARANS yields a local optimum, i.e. a clustering for which no neighbor with a better quality was found. Note that not all neighbors are considered for a local optimum but only *maxneighbor* of them. The parameter *numlocal* allows the user to define the number of these local optima to be searched. In the following, we present the algorithm CLARANS in a C++-like notation. *O*, *k*, *dist, numlocal,* and *maxneighbor* are given as input. The output consists of a set of *k* clusters.

Algorithm CLARANS (int k, function dist, int numlocal, int maxneighbor)

```
for (i = 1; i <= numlocal; i++) {
   current.create_randomly(k);
   j = 1;
   while (j < maxneighbor) {
      current.select_randomly(old, new);
      diff = current.calculate_distance_difference(old, new);
      if (diff < 0) {
         current.exchange(old, new);
         j = 1;}
      else j++; // end if
   } // end while (inner loop)
   dist = current.calculate_total_distance();
   if (dist < smallest_dist) {
      best = current;
      smallest_dist = dist;
   } // end if
} // end for (outer loop)
```

The algorithm assumes the existence of a class `clustering` with methods of the following meaning:

- `create_randomly(k)`
  Creates a random clustering with k clusters, i.e. it selects randomly *k* of the *n* objects as medoids. This selection is random, because it is performed inside the outer loop of the algorithm so that a different selection is required for each iteration.
- `select_randomly(old, new)`
  Selects randomly one of the medoids as `old` and one of the non-medoids as `new`.

- `calculate_distance_difference(old, new)`
  Calculates the difference in total distance between the current clustering and the hypothetical clustering obtained when replacing medoid `old` by `new`. A naive implementation sums up the distance differences for each object implied by the replacement.
- `exchange(old, new)`
  Exchange `old` (a selected medoid) and `new` (a selected non-medoid), i.e. old becomes a non-medoid and new becomes a medoid. Consequently, the assignment of objects to medoids has to be updated.
- `calculate_total_distance()`
  Calculates the total distance for the current clustering.

Note that the algorithm is mainly based on relative distances. No total distance is calculated for the initial clustering of each iteration, and only differences of distances implied by changing medoids are calculated until a local minimum is found. Only then, the total distance is calculated for the current clustering.

Now, we want to analyse the cost of CLARANS, when applied to a database. Our analysis is based on the following assumptions. Let c be the average number of objects stored on one page. The small set of medoids is resident in main memory, while the large set of non-medoids has to reside on disk. The I/O-cost heavily dominates the CPU cost. Therefore, we take the number of disk pages to be read as the cost measure, which is a common approach for database systems. We obtain the following cost for one call of the different methods:

- `create_randomly`
  has to choose k medoids, i.e. it has to read k pages in the worst case.
- `select_randomly`
  has to read just one of the non-medoids, i.e. one page.
- `calculate_distance_difference`
  accesses all objects, i.e. it reads n/c pages.
- `exchange`
  only updates the set of medoids and thus does not access to the disk.
- `calculate_total_distance`
  has to access all objects, i.e. has cost n/c.

The calculation of the number of calls, i.e. the number of iterations in the inner loop, cannot be done in an analytic way because of the heuristic nature of the algorithm. Therefore, we only distinguish whether a method is called within the inner loop (a lot of iterations) or within the outer loop (*numlocal* iterations) of the algorithm.

Thus, the cost of CLARANS is dominated by the cost of the method `calculate_distance_difference`, because its cost is $O(n)$ and it is called inside the inner loop of the algorithm. All other methods are either not as expensive per single call or are not called in the inner loop of the algorithm. A similar observation can be found in [HK 94] stating that in their case the main problem is the efficient computation of the quality of all possible rules. As a consequence of our analysis, in chapter 4 we propose several techniques to improve the efficiency of CLARANS on large databases.

3 An Architecture for Knowledge Discovery in SDBS

[MCP 93] proposes an architecture for a KDD system (cf. figure 1) consisting of the following components: controller, DB interface, focus, pattern extraction, evaluation and knowledge base.

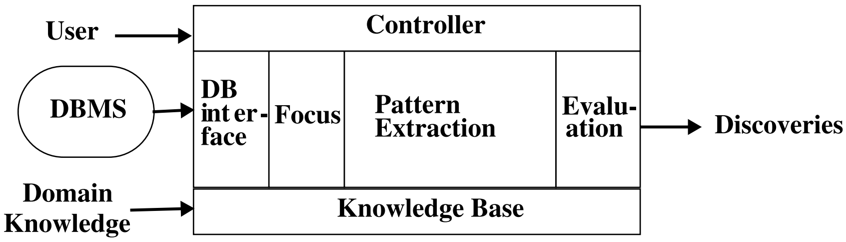

Fig. 1. architecture of a KDD system

So far, we have only considered the task of class identification, a kind of *unsupervised learning*, i.e. learning only from the data without using additional domain knowledge. Thus, we do not require the controller and the knowledge base. In the following, we sketch the components of our KDD system in terms of the above architecture.

The *DB interface* allows the KDD system to select a set of objects from the database fulfilling a given condition on their attributes. Typical queries for SDBS are *region queries* (return all objects from the database intersecting a query polygon) and *nearest neighbor queries* (return the object closest to a query object) [Gue 94]. *Spatial access methods* (SAM) have been developed to support efficient processing of such queries (cf. [BHKS 93] [BKSS 94]). An *approximation* is a simple object with limited complexity preserving the main properties of a complex spatial object. The most common approximation is the *bounding box* (BB), i.e. the minimal rectilinear rectangle containing a given spatial object. Therefore, most SAMs are designed to manage rectangles.

The *R*-tree* [BKSS 90] is a SAM which is very efficient for points and rectangles. Each node of the tree represents a *page*, the unit of transfer from secondary storage to main memory. Therefore, the number of rectangles per node is constrained by a lower and an upper limit, such that a high storage utilization is obtained and consequently the number of disk pages to be read for query processing is as small as possible. The nodes storing the data objects are called *data pages*, the nodes organizing the directory are called *directory pages*. As soon as overlapping data rectangles do not fit into the same page, an overlap of directory pages will occur. This overlap of directory pages has to be minimized for efficient query processing. Therefore, the R*-tree uses a heuristic splitting strategy when the capacity of a page is exceeded after the insertion of a new object.

The *focusing component* determines which parts of the database are relevant for pattern extraction. In relational DBS, e.g. one could focus on some attributes of the tuples yielding most information or focus on a randomly drawn sample of all tuples. Finally, the focusing component asks queries to the DB interface obtaining the input for pattern extraction. The focusing component is outlined in chapter 4.

Pattern Extraction is based on the data returned by the focusing component. In general, pattern extraction is a multi-step process, i.e. a cluster might be input for another step of cluster analysis. For example, a first step might cluster the data according to a non-spatial attribute like landuse, and a second step would cluster all data within one of the resulting clusters according to the spatial attributes.

The *evaluation component* should determine the statistical significance of the extracted patterns and support an application-specific evaluation of their usefulness by the user. So far, we do not compute the significance. The clusters extracted are visualized to the user in a graphical way, either all clusters at the same time or one cluster at a time together with its neighboring clusters.

4 The Focusing Component

In chapter 2, we have concluded that the most expensive operation of CLARANS is calculating the difference of the total distances of two clusterings. There are two approaches to improve the efficiency of this operation. First, a reduction of the number n of all objects will result in a significant speed-up, because the calculation of the distance difference is $O(n)$. Second, a careful analysis shows that actually not all n objects contribute to the result of the operation, so that efficiency can be improved by restricting the access to the relevant objects. In this chapter, we present three different focusing techniques exploiting both approaches.

4.1 Focus on Representative Objects

The number of all possible clusterings of a database depends on n and k. In order to reduce the time complexity, we propose to apply CLARANS not to the whole database, but to select a relatively small number of representatives from the database and to apply CLARANS only to these representatives. This is a kind of *sampling*, a technique common in KDD systems, e.g. [KR 90]. The quality of the sampling is crucial for the quality of the resulting clustering. Our DB interface supports a new way of selecting representatives from a SDBS. From each data page of the R*-tree, we select the most central object as a representative. Thus, the clustering strategy of the R*-tree, which minimizes the overlap between directory rectangles, yields a well-distributed set of representatives.

Let the *center of the data page* be the center of the bounding box of its objects. The *most central object* in a data page is the object with the minimal distance of its center from the center of the data page. Figure 2 illustrates the selection of representatives according to this definition for some data pages.

An obvious question is, whether it is reasonable to let the R*-tree perform the whole clustering in one step without using CLARANS in a second step. The answer is “no” because of the following reasons:

- The R*-tree does not allow the users to specify the number k of clusters, it derives k indirectly from n and from the capacity of a page. This k may be inappropriate for a given application and may yield clusterings with a high total distance.
- All clusters (i.e. the directory rectangles) have a rectangular shape and, furthermore, these rectangles have to be parallel to the axes of the coordinate system.

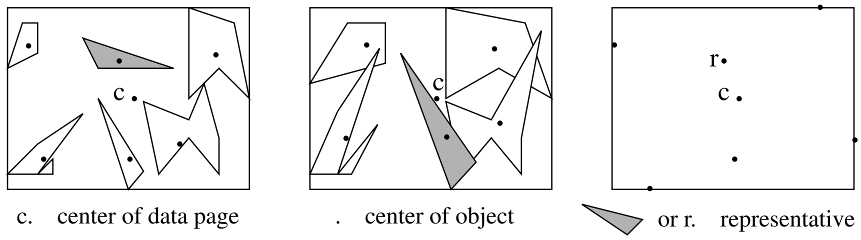

Fig. 2. example data pages of an R*-tree with representatives selected

We propose to combine the good properties of the R*-tree and of CLARANS in the following two-step approach:

1.) Extract one representative for each data page of the R*-tree.
2.) Cluster the representatives using CLARANS and return *k* medoids.

For the purpose of extracting the representatives, we need a new query, called *centroid query*, returning the most central object for each data page. This query requires a scan over all data pages of an R*-tree, i.e. *n/c* pages have to be read. On the other hand, when focusing on representatives, CLARANS only has to cluster *n/c* objects instead of *n* objects.

4.2 Focus on Relevant Clusters

In this section, we take a closer look at the calculation of the difference of total distance between two neighbor clusterings, i.e. clusterings differing in exactly one medoid. This method gives the main contribution to the cost of CLARANS. The algorithm presented by [NH 94] performs a loop over all non-medoid objects for calculating the difference of distance. This is prohibitive when working on a database, because all these objects would have to be loaded into main memory. Therefore, our goal is to restrict the calculation to the relevant parts of the database.

There are four different cases of non-medoid objects *o* to be distinguished when calculating the distance difference between a current clustering and the resulting clustering after exchanging a medoid *old* with a non-medoid *new*. They are illustrated in the following figures by examples of 2D-points.

1.) The current medoid of *o* is *old,* and o is closer to its second closest medoid than to *new*. Then *o* will be inserted into the cluster whose medoid is its second closest medoid (see figure 3).

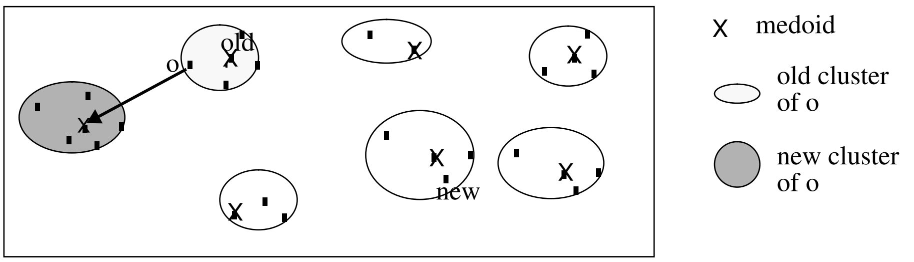

Fig. 3. Example of Case 1

2.) The current medoid of *o* is *old,* and o is closer to *new* than to its second closest medoid. Then *o* will be inserted into the cluster whose medoid is *new* (see figure 4*)*.

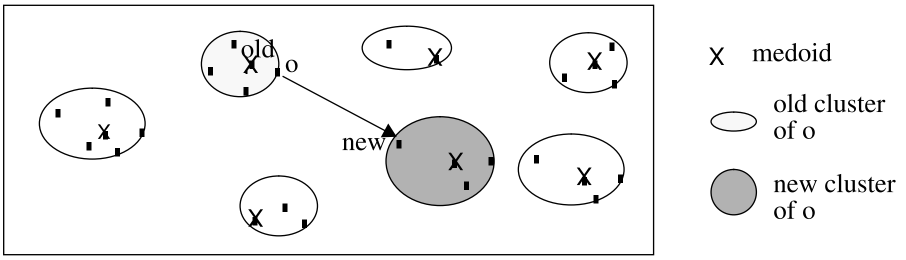

Fig. 4. Example of Case 2

3.) The current medoid of *o* is different from *old* and o is closer to its medoid than to *new*. Then *o* will stay in its cluster (see figure 5).

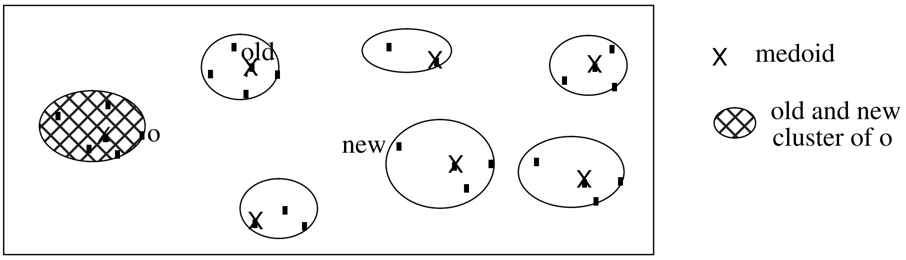

Fig. 5. Example of Case 3

4.) The current medoid of *o* is different from *old*, and o is closer to *new* than to its current medoid. Then *o* will be inserted into the cluster whose medoid is *new* (see figure 6).

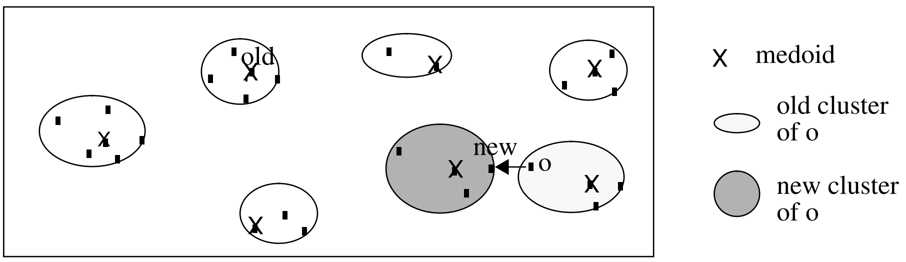

Fig. 6. Example of Case 4

Only in cases 1.), 2.) and 4.) *o* will be moved to another cluster, i.e. only in these cases the total distance for *o* will change. The objects of case 1.) form a subset of the objects with medoid *old*, cases 2.) and 4.) cover the objects with medoid *new*. Thus, only the objects belonging to the clusters of *old* and *new* contribute to the distance difference. Instead of reading all non-medoid objects from disk (i.e. the objects of all k clusters), we just have to read the objects of two clusters. Assuming the same average size for all clusters, we expect a performance gain of $k/2$ compared to [NH 94]. However, we need an efficient way of retrieving exactly the objects of a given cluster from the database. This is the issue of section 4.3.

4.3 Focus on a Cluster

In this section, we will discuss how to retrieve all the objects of the cluster for a given medoid efficiently from the database. A naive solution of this problem will calculate all distances $dist(o,m)$, for all $o \in O$ and all $m \in M$. This technique would require n/c pages to be read from disk.

Now, we want to solve this problem more efficiently. According to chapter 2, the distance of polyhedron objects is defined by using the distance of points. Thus, in the following, we only consider point objects without loss of generality. We construct a polyhedron for a medoid m_i such that all objects within this polyhedron belong to the cluster with medoid m_i while no objects from other clusters are contained in it. Then, we retrieve the objects whose centers intersect this polyhedron by a region query. The construction of this polyhedron can efficiently be performed, because it needs only the medoids and the bounding box of all objects in the database, but not all objects in the database. Assuming the same average size for all clusters, we expect only $n/k*c$ instead of n/c pages to be read from disk with this focusing technique.

To construct the polyhedron, we need the following definitions. Given two different medoids $m_i, m_j \in M$, the *perpendicular bisector* of m_i and m_j is defined by (1). Obviously, the bisector is a plane bounding the *half-space* H_{ij}, which is defined by (2).

$$B_{ij} := \{x \in \mathrm{R}^3 \mid dist\,(x, m_i) = dist\,(x, m_j)\} \qquad (1)$$

$$H_{ij} := \{x \in \mathrm{R}^3 \mid dist\,(x, m_i) \le dist\,(x, m_j)\} \qquad (2)$$

For any medoids m_i and m_j, all objects closer to m_i than to m_j are located in the half-space H_{ij}. Let $V(i)$ denote the intersection of the k-1 half-spaces, i.e. $V(i) = \bigcap H_{ij}$. $V(i)$ contains all the objects which are closer to m_i than to any other medoid of set M, i.e. $V(i)$ contains all objects of the cluster with medoid m_i. At the same time, $V(i)$ contains no object of other clusters. Otherwise, this object would be contained in one of the half-spaces H_{ij}. This is a contradiction to definition (2). $V(i)$ is called the *Voronoi polyhedron* (or polygon in the 2D case) associated with m_i [PS 85]. $V(i)$ may be bounded or unbounded. If it is bounded, we use $V(i)$ as a query region to the R*-tree. Otherwise, we use the intersection of $V(i)$ with the bounding box (BB) of all objects of O as a query region. The bounding box can be easily computed without accessing all the objects of the database only by using the root of the R*-tree. Figure 7 illustrates the constructed query region for a small database of 2D-polygons.

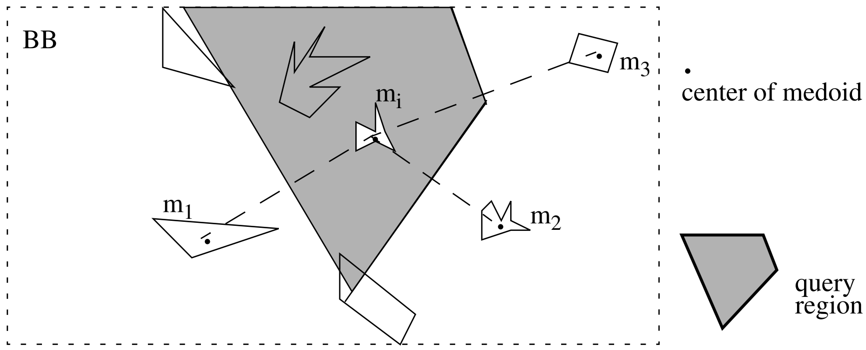

Fig. 7. The query region for all objects of the cluster with medoid m_i

5 Application and Performance Evaluation

We apply the proposed clustering techniques to a large protein database and evaluate their performance in this context. We introduce the protein database (section 5.1), illustrate the KDD task in this application (section 5.2) and evaluate focusing on representatives with respect to effectiveness and efficiency (section 5.3).

5.1 BIOWEPRO - a SDBS for Protein-Protein Docking

Proteins are biomolecules consisting of some hundreds to some thousands of atoms. Their mode of operation lies in the interaction with other biomolecules, for example proteins, DNA or smaller partner molecules. These interactions are performed by connecting the partner molecules, and are therefore called *docking*.

Molecular biologists point out that the geometry of the molecular surfaces at the interaction site plays an important role, along with the physicochemical properties of the molecules. A necessary condition for protein-protein docking is the complementarity of the interaction site with respect to surface shape, electrostatic potential, hydrophobicity etc. Therefore, a database system for protein-protein docking has to process queries for proteins with similar or complementary surfaces.

In the BIOWEPRO (Biomolecular Interactions of Proteins) project (cf. [EKSX 95], [SK 95]) we are developing a SDBS to support protein-protein docking. We use the crystallographically determined atom coordinates of proteins and protein complexes from the Brookhaven Protein Data Bank ([Ber 77], [PDB 94]), presently containing some 3,000 proteins. Each protein has a triangulated surface with some 10,000 3D points. For each point on the protein surface, several geometric and physicochemical features are computed. The *solid angle (SA),* e.g., [Con 86] is a geometric feature describing the degree of convexity or concavity of the surface in the neighborhood of the considered point.

5.2 KDD in the BIOWEPRO Database

The search for similar protein surfaces is not performed at the level of surface points, but at the level of surface segments, resulting in a significant reduction of the number of both, the objects in the database and the answers to a given query. A *segment* is defined as a set of neighboring surface points with similar non-spatial attributes, e.g. with similar SA values. The segments should have a good correlation with the known docking sites of the proteins, i.e. a docking site on a protein surface should consist of a small number of segments. Thus, the KDD task is to find a segmentation of protein surfaces supporting the processing of docking queries. There are two possible ways to combine the processing of spatial and non-spatial attributes [NH 94]:

1.) *Spatial dominated approach*. Apply the clustering algorithm first on the spatial attribute to obtain segments. The number of clusters is determined heuristically by using the number of local extrema of SA. Then generalize the non-spatial attributes for all 3D points of a given segment to classify the shape of the segment.

2.) *Non-spatial dominated approach*. Apply the clustering algorithm first on the non-spatial attributes with the number of clusters, e.g., set to 5. In the second step, we cluster each of the 5 non-spatial clusters using the spatial attributes such that these clusters are split into segments of neighboring points.

For an illustration, we sketch the application of the spatial dominated approach in the BIOWEPRO database. Figure 8 depicts some of the segments found on the surface of protein 2ptc by spatial clustering.

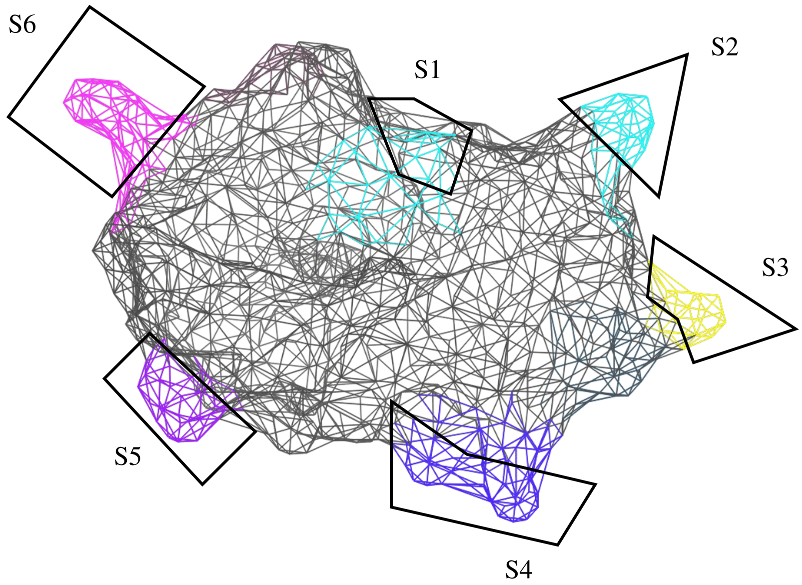

Fig. 8. Example segments on the surface of protein 2ptc

Based on the values of SA, three classes of shapes (convex, neutral and concave) can be distinguished. Since no crisp definitions of these classes are available, we perform fuzzy classification. Figure 9 presents our fuzzy membership functions for the three classes.

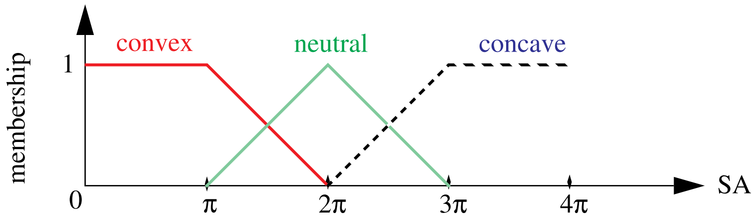

Fig. 9. Fuzzy membership functions for the three classes of shape

According to the membership functions, segment S1 is neutral (with grade 0.826) and convex (with grade 0.174), whereas S2 is convex (with grade 0.981) and neutral (with grade 0.019).

5.3 Evaluation of Focusing on Representatives

In this section, we present experimental results from the BIOWEPRO database evaluating the technique of focusing on representatives with respect to efficiency and effectiveness. Our measure of effectiveness is the average distance of the resulting clusterings, i.e. the average distance of an object from its medoid. Efficiency is measured by the CPU runtime of the whole processing. All experiments have been run on an HP 9000/735 workstation.

We use the protein hemoglobin (4hhb) for our experiments, because it is one of the largest objects in the database. The surface of 4hhb consists of 50,559 points, and for each of these points we store the 3D-coordinates along with the value of SA. The number of clusters is set to 10 and *numlocal* set to 2. In the first set of experiments, we directly apply CLARANS on 4hhb with *maxneighbor* varying from 250 over 500 to 1,000. In the second set of experiments, we use “focusing on representatives”. We obtain 1,027 representatives out of the 50,559 points for 4hhb, and CLARANS is then applied to this set of representatives.

The results on effectiveness in terms of the average distance are presented in figure 10. Using the focusing technique, we observe a decrease of effectiveness ranging

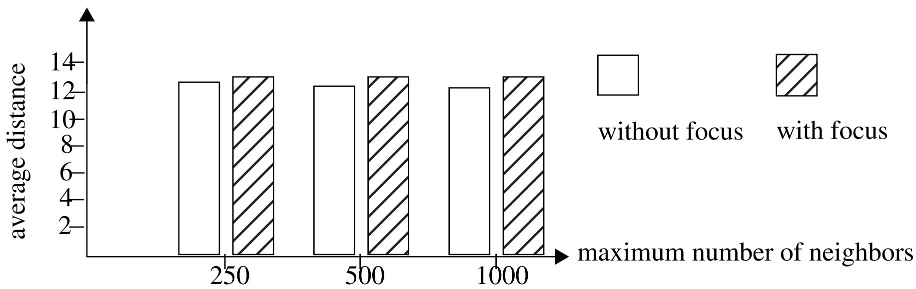

Fig. 10. Comparison of Effectiveness

from 1.5% to 3.2% compared to clustering without focus. Figure 11 depicts the results of the comparison of efficiency. Focusing improves the efficiency by a factor ranging from 48 to 158 in comparison to clustering without focus.

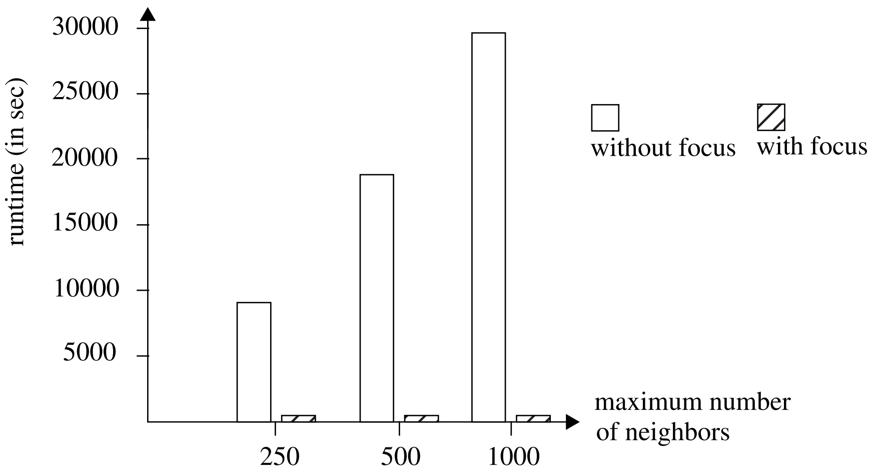

Fig. 11. Comparison of Efficiency

To conclude, focusing improves efficiency by a factor of 48 to 158, whereas the loss of effectiveness is only 1.5% to 3.2%.

6 Conclusion

We use the clustering algorithm CLARANS [NH 94] for class identification in SDBS. Our analysis points out that the most expensive operation of CLARANS, when applied to a large SDB, is calculating the difference of the total distances of two clusterings.

The DB interface of our KDD system is based on the R*-tree, a well-known spatial access method. For the purpose of focusing, we need a new query, called centroid query, returning the most central object for each data page.

The focusing component supports three types of focusing. Focusing on representative objects significantly reduces the number of objects to be clustered. Focusing on relevant clusters restricts the calculation of the difference of distance between two clusterings to the relevant clusters, i.e. to the cluster of the medoid and the cluster of the non-medoid to be exchanged. Focusing on a given cluster is performed by determining the minimum polyhedron intersecting all objects of this cluster.

We have applied the proposed clustering techniques to real data from a large protein database used for predicting protein-protein docking. There are other types of applications for the proposed clustering method, because its only requirement is the availability of an appropriate distance function for any two objects. In [NH 94] CLARANS is applied in the context of a geographic information system. We want to apply our approach to a k-dimensional feature space, in which CAD parts are described by k non-spatial features. The first goal is to find classes of CAD parts based on their features. Furthermore, we want to discover associations between the different features, indicating e.g. the redundancy of a given feature.

We have performed an evaluation of focusing on representatives on the protein database. In terms of efficiency, CLARANS with focusing outperforms CLARANS by a factor of 48 to 158. The decrease of effectiveness, using the focus on representatives, is only 1.5% to 3.2% compared to CLARANS. Thus, focusing on representatives offers a very good trade off between efficiency and effectivity.

Future research will have to consider the following issues. CLARANS randomly selects two objects to be exchanged and does not consider any alternatives if the exchange results in a reduction of the total distance of the clustering. Heuristic strategies for selection should reduce the huge size of the search space and thus improve the efficiency of pattern extraction. So far, we have created crisp clusterings, i.e. each object has been assigned to a unique cluster. However, due to the spatial nature of the objects, it is possible that an object intersects the area of two clusters at the same time. A similar situation occurs when two objects have the same distance from two different medoids. In both cases, fuzzy clustering techniques, assigning an object to several clusters with varying degrees of membership, seem to be more appropriate than crisp clustering methods. We intend to explore them in our future work.

Acknowledgment

We thank Thomas Seidl for engaging and fruitful discussions on the subject of this paper and for his support in the performance evaluation on the BIOWEPRO data.

References

[AIS 93] Agrawal R., Imielinski T., Swami A.: *"Database Mining: A Performance Perspective"*, IEEE Transactions on Knowledge and Data Engineering, Vol.5, No.6, 1993, pp. 914-925.

[Ber 77] Bernstein F. C., Koetzle T. F., Williams G. J., Meyer E. F., Brice M. D., Rodgers J. R., Kennard O., Shimanovichi T., Tasumi M.: *'The Protein Data Bank: a Computer-based Archival File for Macromolecular Structures',* Journal of Molecular Biology, Vol. 112, 1977, pp. 535-542.

[BHKS 93] Brinkhoff T., Horn H., Kriegel H.-P., Schneider R.: *'A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems',* Proc. 3rd Int. Symp. on Large Spatial Databases, Singapore, 1993, Lecture Notes in Computer Science, Vol. 692, Springer, pp. 357-376.

[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, 1990, pp. 322-331.

[BKSS 94] Brinkhoff T., Kriegel H.-P., Schneider R., Seeger B.: *'Efficient Multi-Step Processing of Spatial Joins',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Minneapolis, MN, 1994, pp. 197-208.

[Con 86] Connolly M. L.: *'Measurement of protein surface shape by solid angles'*, Journal of Molecular Graphics, Vol. 4, No. 1, 1986, pp. 3-6.

[EKSX 95] Ester M., Kriegel H.-P., Seidl T., Xu X.: "Shape-based Retrieval of Complementary 3D Surfaces in Protein Databases", (in German), Proc. GI Conf. on Database Systems for Office Automation, Engineering, and Scientific Applications.1995, Berlin: Springer 1995.

[FPM 91] Frawley W.J., Piatetsky-Shapiro G., Matheus J.: *"Knowledge Discovery in Databases: An Overview"*, in: Knowledge Discovery in Databases, AAAI Press, Menlo Park, 1991, pp. 1-27.

[Gue 94] Gueting R.H.: *"An Introduction to Spatial Database Systems"*, Special Issue on Spatial Database Systems of the VLDB Journal, Vol.3, No.4, October 1994.

[HCC 93] Han J., Cai Y., Cercone N.: *"Data-driven Discovery of Quantitative Rules in Relational Databases"*, IEEE Transactions on Knowledge and Data Engineering, Vol.5, No.1, 1993, pp. 29-40.

[HK 94] Holsheimer M., Kersten M.L.: *"Architectural Support for Data Mining"*, Proc. AAAI Workshop on Knowledge Discovery in Databases, Seattle, Washington, 1994, pp. 217-228

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

[LHO 93] Lu W., Han J., Ooi B.C.: *"Discovery of General Knowledge in Large Spatial Databases*", Proc. Far East Workshop on Geographic Information Systems, Singapore, 1993, pp. 275-289.

[MCP 93] Matheus C.J., Chan P.K., Piatetsky-Shapiro G.: *"Systems for Knowledge Discovery in Databases"*, IEEE Transactions on Knowledge and Data Engineering, Vol.5, No.6, 1993, pp. 903-913.

[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, 1994, pp. 144-155.

[PDB 94] Protein Data Bank: *'Quarterly Newsletter No. 70 (Oct. 1994)',* Brookhaven National Laboratory, Upton, NY, 1994.

[PS 85] Preparata F. P., Shamos M. I.: "*Computational Geometry*", Springer 1985.

[SK 95] Seidl T., Kriegel H.-P.: *'Solvent Accessible Surface Representation in a Database System for Protein Docking',* Proc. 3rd Int. Conference on Intelligent Systems for Molecular Biology (ISMB–95), Cambridge, UK, AAAI Press, 1995.