AEC Algorithm: A Heuristic Approach to Calculating Density-Based Clustering *Eps* Parameter

Marcin Gorawski and Rafal Malczok

Silesian University of Technology,
Institute of Computer Science,
Akademicka 16,
44-100 Gliwice, Poland
{Marcin.Gorawski, Rafal.Malczok}@polsl.pl

Abstract. Spatial information processing is an active research field in database technology. Spatial databases store information about the position of individual objects in space [6]. Our current research is focused on providing an efficient caching structure for a telemetric data warehouse. We perform spatial objects clustering when creating levels of the structure. For this purpose we employ a density-based clustering algorithm. The algorithm requires an user-defined parameter *Eps*. As we cannot get the *Eps* from user for every level of the structure we propose a heuristic approach for calculating the *Eps* parameter. Automatic *Eps* Calculation (AEC) algorithm analyzes pairs of points defining two quantities: distance between the points and density of the stripe between the points. In this paper we describe in detail the algorithm operation and interpretation of the results. The AEC algorithm was implemented in one centralized and two distributed versions. Included test results present the algorithm correctness and efficiency against various datasets.

1 Introduction

Many computer research areas require spatial data processing. Computer systems are used for gathering and analyzing information about traffic in big cities and highways. The systems utilize drivers' cell phones signals to track vehicles. Stored tracking data is then analyzed and used to support the process of making decisions such as building new bypasses, highways and introducing other rationalizations. More and more people are interested in on-line services providing very precise and high-quality maps created from satellite images (an example can be found at [3]). There are more very interesting projects concerning spatial data processing; for details please refer to [1,2].

Another very important branch of spatial systems is telemetry. We work on a telemetric system of integrated meter readings. The system consist of utility meters, collecting nodes and telemetric servers. The meters are located in blocks of flats, housing developments etc. They measure water, natural gas and energy usage and send the readings to the collecting nodes via radio. The collecting nodes

T. Yakhno and E. Neuhold (Eds.): ADVIS 2006, LNCS 4243, pp. 90–99, 2006.

collect the readings and send them to the telemetric servers through the Ethernet network. The data from the telemetric servers are extracted, transformed and then loaded to the database of the data warehouse. Apart from meter readings, the data warehouse database stores information about the meters' geographical location and their attributes.

The remaining part of the paper is organized as follows. In the next subsection we present our motivation and describe the problem we are trying to solve. We then provide all the details of the proposed solution. We also include test results which present the algorithm efficiency and correctness against various datasets. We conclude the paper presenting our future plans.

1.1 Problem Description

The most typical use for the presented telemetric data warehouse is to investigate utilities consumption. Our current research is focused on providing fast and accurate answers to spatial aggregating queries. We are in the process of designing and implementing a hierarchical caching structure dedicated to telemetry-specific data. We named the structure a Clustered Hybrid aR-Tree (CHR-Tree) because we intend to use clustering to create the structure nodes, and, like in the aR-Tree [6], the structure nodes store aggregates.

We already have a solution to a problem of storing and processing the aggregates in the CHR-Tree nodes [5]. Currently we are trying to construct the structure of the CHR-Tree. To create the intermediate level nodes we employ density-based clustering algorithm. We decided to use the DBRS algorithm [7]. Although efficient and scalable, the algorithm requires an user-defined parameter *Eps*. *Eps* is a parameter defining a half of the range query square side. The side length is used by the clustering algorithm to evaluate range queries when searching for neighboring points. To the best of our knowledge, there is no automatic method for calculating the *Eps* parameter for the density-based clustering. Authors of the DBScan algorithm proposed in [4] a simple heuristics to determine the *Eps* parameter. However, the heuristics cannot be considered automatic as it requires user interaction. As we cannot get the *Eps* parameter from the user for every level of the structure, we propose an empirical Automatic *Eps* Calculation (AEC) algorithm. The algorithm is not limited to the telemetry-specific data and can be applied to any set of points located in two-dimensional space.

2 Automatic *Eps* Calculation Algorithm

The AEC algorithm investigates a distribution of the points in a given dataset. The analyzed datasets may be large, hence we have to limit the amount of processed data. We decided to use a random sampling approach because it can give good results in acceptable time. The AEC algorithm uses the following sets of data:

- a set of all points P. The points in the set P are located in some abstract region, in two-dimensional space,

- a set N. The set contains points randomly chosen from the P set. There is a function $createSetN()$ that is used for creating the N set. The function takes one optional parameter r, that defines the region from which the points are being picked. When the r parameter is present during the N set generation, we mark the set with an appropriate subscript: N_r,
- a set H. Like the N set, the H set contains points randomly picked from the P set. In the case of the H set, the function creating the set is named $createSetH()$. Next to the r optional parameter, whose meaning is identical as for the N set, the $createSetH$ function takes another parameter defining the point that is skipped during random points drawing. The H sets are created for points from the N set. The notation H_{r,n_i} means that the H set was created for the point $n_i \in N$; the point n_i was skipped during random points drawing and the points in H are located in a region r.

The cardinalities of N and H sets are the AEC algorithm parameters. Thanks to the parametrization of those values we can easily control the precision and the algorithm operation time. The cardinality of the N set is defined as the percent of the whole P set. The cardinality of the H set is defined directly as the number of points creating the set.

2.1 Algorithm Coefficients

The first step of the AEC algorithm operation is to pick randomly from the P set points creating the set N. In the next step, for each point $n_i \in N$ the algorithm creates set H_{n_i}. Utilizing the created sets, the algorithm evaluates three coefficients.

The first calculated coefficient is the Euclidean distance between the point n_i and point from the related H_{n_i} set. The distances are calculated for all points in the N set and all related H sets.

However, knowing only the distance between points p_i and p_j is not enough to estimate the Eps parameter. Missing is the knowledge about the neighborhood of the analyzed points; actually about the points in the region between the investigated points p_i and p_j. We introduced a coefficient PIS (Points In Stripe). The value of $PIS(p_i, p_j)$ is the number of points located in a *stripe* connecting the points p_i and p_j.

To evaluate the PIS coefficient value for a pair of points we use one spatial query and four straight lines equations. Having the p_i and p_j points coordinates we can easily calculate the parameters a and b of the straight line L equation $y = ax+b$. The line L contains the points p_i and p_j. In the next step we calculate equations of the lines perpendicular to L in points p_i and p_j, respectively L_{p_i} and L_{p_j} (we do not include the equations because of the complicated notation and straightforward calculations). The final step is to calculate two lines parallel to L, the first above line $L - L_a$ and the second below line $L - L_b$. The distance between the parallel lines and the L line (the difference in the b line equation coefficient) is defined as a fraction of the distance between points p_i and p_j. The fraction is the AEC algorithm parameter named $stripeWidth$; $stripeWidth \in (0, 1)$. The

lines create a *stripe* between the points, and the *stripe* encompasses some number of points.

Having the lines equations we can easily calculate, whether or not an arbitrary point from the set P is located inside the stripe between points p_i and p_j. In order to reduce the number of points being analyzed we evaluate a rectangle encompassing the whole stripe. The rectangle vertexes coordinates are set by calculating the coordinates of the points where the stripe-constructing lines (L_a, L_b, L_{p_i} and L_{p_j}) cross, and then choosing the extreme crossing points coordinates. Using the stripe-encompassing rectangle we execute the range query to choose the points which can possibly be located within the stripe between p_i and p_j. In the next step, only the points chosen by the range query are examined if they are located within the stripe.

After calculation of the PIS coefficient we are equipped with two values, which provide interesting knowledge not only about distance between points p_i and p_j but also about their neighborhood. Basing on the distance between points: $dist(p_i, p_j)$ and the number of points in a stripe between points $PIS(p_i, p_j)$ we can calculate another coefficient, which is a density of the stripe between p_i and p_j: $dens(p_i, p_j) = \frac{PIS(p_i,p_j)}{dist(p_i,p_j)^2 \cdot stripeWidth}$.

Figure 1 presents an example of a stripe between two points. The $stripeWidth$ parameter was set to 0.98. In this example we are checking two pairs of points: p_5, p_8 and p_3, p_6. We used a dashed line to indicate the line linking two points. Solid lines depict the parallel and perpendicular lines. Rectangles drawn with spotted lines describe the regions encompassing the stripes. From the picture we see, that there is one point between points p_5, p_8 and there are 3 points between points p_3, p_6.

The distances (in millimeters) between the points p_5, p_8: dist(5, 8) = 302.1, and p_3, p_6: dist(3, 6) = 79.2. The density of the stripe between p_5 and p_8 is $dens(p_5, p_8) = \frac{1}{302.1^2 \cdot 0.98} = 0.11 \cdot 10^{-4}$ and for p_3, p_6: $dens(p_3, p_6) = \frac{3}{79.2^2 \cdot 0.98} = 4.88 \cdot 10^{-4}$. From the example we see that the density inside the cluster is much greater than outside the cluster. The density coupled with the distance between points brings much more knowledge than the distance itself. Now we are able to deduce whether two points are relatively close to each other, and whether they are located in a dense neighborhood or, on the other hand, whether the points are relatively distant and there are almost no points in the stripe between them. After analyzing the operation of the density-based algorithms, that is executing a series of range queries, we decided to search not for a distance between points in clusters or for the thinnest cluster diameter, but rather for a minimal distance between clusters. A value based on the minimal distance can be used as the *Eps* parameter in the density-based clustering algorithm. Using a minimal distance between clusters as the *Eps* parameter should result in grouping all the points whose distances to their closest neighbors are shorter than the minimal distance between clusters (they are in one cluster) and not grouping points when the distance between them is greater than the minimal distance between clusters.

2.2 Algorithm Operation

The AEC algorithm operates in iterative mode. In every iteration the algorithm tries to minimize the calculated minimal distance between clusters. Below we present the pseudocode of the algorithm.

```
 (1) CalculateEps(maxIter, maxRptd, distInit, densInit) : Float
 (2)   distCur := distInit; densCur := densInit; // initialize variables
 (3)   distPrev := distCur;  // stores previously calculated distance
 (4)   iter := 0; rptd := 0; // # of iterations and # of repeated results
 (5)   WHILE iter < maxIter AND rptd < maxRptd DO
 (6)     N := createSetN(); // create set N
 (7)     FOR ni IN N DO // for every point ni in N do
 (8)       niDist := Float.MAX_VALUE; // initialize results for point ni
 (9)       niDens := Float.MAX_VALUE;
(10)       rni := createRect(distCur, ni); // create rectangle for point ni
(11)       rniDens := getAvgDens(rni); // calculate rectangle density
(12)       Hni := createSetH(ni, rni); // create set H for point ni
(13)       FOR hj IN Hni DO
(14)         tDist := calcDist(ni, hj); // calculate distance
(15)         tPIS  := calcPIS(ni, hj);  // calculate # of points in stripe
(16)         tDens := calcDens(tDist, tPIS); // calculate stripe density
(17)         IF tPIS > 0 AND tDist < niDist AND tDens <= rniDens THEN
(18)           niDist := tDist; // set new results for point ni
(19)           niDens := tDens;
(20)         END IF;
(21)       END FOR; // loop for points from Hni set
(22)       IF niDist < distCur AND niDens <= densCur THEN
(23)         distCur := niDist; // update iteration results
(24)         densCur := niDens;
(25)       ELSE
(26)         IF niDist < distCur THEN // check suspected region
(27)           sDist, sDens := checkSuspReg(ni, hj);
(28)           IF sDist < distCur AND sDens <= densCur THEN
(29)             distCur := sDist; // update iteration results
(30)             densCur := sDens;
(31)           END IF;
(32)         END IF; // suspected region condition
(33)       END IF; // updating results condition
(34)       IF distCur = distPrev THEN
(35)         rptd := rptd + 1; // increase number of repeated results
(36)       ELSE
(37)         rptd := 0;
(38)         distPrev := distCur; // store previously calculated value
(39)       END IF; // repeated result condition
(40)       iter := iter + 1; // increase number of performed iterations
(41)     END FOR; // loop for points from N set
(42)   END WHILE; // main loop
(43)   RETURN distCur;
(44) END; // CalculateEps
```

The input parameters are: the maximal number of iterations, the maximal number of repeated results and the initial distance and density. Two first values are used for creating the breaking condition (line 5). The iterations are broken if the number of performed iterations is greater than the maximal number of iterations or the result returned by consecutive iterations was repeated a given number of times. Next two parameters, the initial values, should be set in a way that they reduce the number of iterations to minimum, but do not narrow down the set of possible solutions. We use an average distance between random points, and average density related to the distance as the initial values.

In the iterative section the AEC algorithm first creates the N set (line 6). Then, for every point $n_i \in N$ performs the following:

- creates the r_{n_i} rectangle and calculates its density (lines 10,11). The r_{n_i} rectangle has its center in the n_i point and its sides are $2 \times distCur$ length,
- creates $H_{r_{n_i},n_i}$ set (line 12). Then for every point $h_j \in H_{r_{n_i},n_i}$ the algorithm calculates the distance and the density of the stripe between the n_i and h_j points (lines 14-16).
- from all the results the algorithm chooses the best distance and density pair (lines 17-20) for the n_i point,
- condition in line 22 chooses the best distance and density pair for the whole N set,
- if the results for the n_i point satisfies only the first part of the condition (line 26), the algorithm checks a *suspected region* (see below),
- in line 34 the algorithm checks if the returned result is repeated,
- as the result the function returns the calculated $distCur$ value.

Suspected Region. The case of a *suspected region* is considered for points p_i, p_j when only the distance condition ($dist(p_i, p_j) < dist_{cur}$)) holds, the density condition ($dens(p_i, p_j) <= dens_{cur}$)) does not. Our experiments show that there are two possible scenarios resulting in examining a *suspected region*:

1. the points p_i, p_j are located close to each other inside a cluster. The distance then is short, but the density of the stripe between the points is high (the scenario is marked as rectangle (1) in fig. 2).
2. the points p_i, p_j are located in separate clusters but they are not border points (according to the definition presented in [7]). The situation is presented in fig. 2 as rectangle (2). The density of the stripe between the points is increased by the presence of the border points of both clusters.

Of considerable interest is the second case. The AEC algorithm does not analyze distances with the zero PIS coefficient. There are many cases when the clusters' shapes make it difficult to randomly pick two points so that one of them is a border point of the first cluster and the second is located near the border of the second cluster. The analysis of the *suspected region* is performed as follows:

1. define the *suspected region*. The rectangle r_s for the *suspected region* has its center directly between the points p_i and p_j (fig. 2). In the next step calculate the density $dens_{r_s}$ of the r_s.
2. create a set of points N_{r_s}.
3. for each point $n_i \in N_{r_s}$ create a set H_{n_i,r_s}, then calculate distances and densities of the stripe between points n_i and the related points $h_j \in H_{n_i,r_s}$. As the result choose the minimal distance with the minimal density.

In the event the calculated result density is less than the average density of the r_s region, the *suspected region* analysis results are compared with the results of the analysis in the iterative section of the AEC algorithm. For a pair of points located inside a cluster the *suspected region* analysis does not influence the results because the density condition is not satisfied (the density is high inside a cluster). But for the points located in two different clusters the analysis often gives important results.

The amount of points checked during *suspected regions* analysis depends on the number of points in the r_s rectangle. If the number is less than the N set cardinality, then all the points are checked. But if the number is greater, the cardinality of the N_{r_s} set equals the cardinality of the N set created in the iterative section of the algorithm. The situation is identical for the H sets.

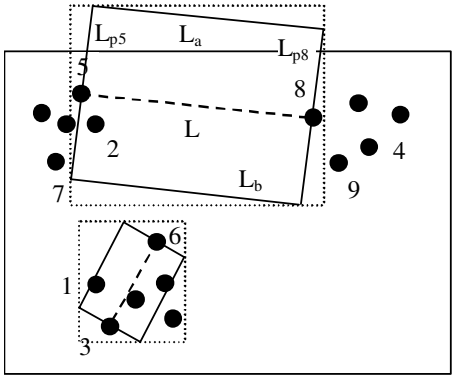

Fig. 1. Hypothetical stripes between two pairs of points

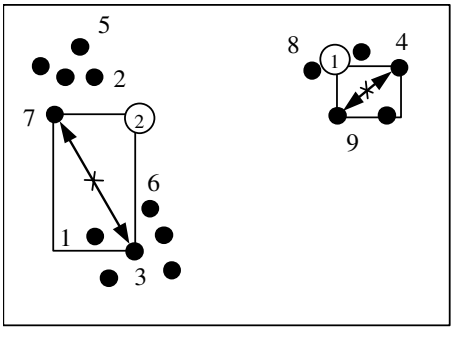

Fig. 2. The *suspected regions* are being analyzed with special attention

2.3 Implementation

In order to improve the efficiency of the AEC algorithm we used distributed processing. The structure of the distributed system consist of a client and a few servers. Each server stores the set of all points P and each server performs the same operations but for different subsets of points. Each server is assigned a set of points from which it creates the N sets. The sets are disjoint for all servers. Thus we minimize the possibility that some servers examine the same pair of points. The H sets are created from the whole P set, with no limitations.

We implemented two different distributed versions of the AEC algorithm. The first version named *at once* (AO) assumes, that client and servers do not communicate during the process of *Eps* evaluation. The servers calculate the minimal distance between clusters with the lowest density and return the results to the client which selects the best result (the shortest distance with the lowest related density). Disadvantage of this approach is that the servers calculations are less precise because they use N sets which cardinalities are only $\frac{1}{K}$ of cardinalities of the sets used in the centralized version (where K is the number of servers). The second version named iterative (IT) assumes that the client requests the servers to perform the i^{th} iteration of the whole process. The servers return results of the i^{th} iteration to the client. The client selects the best result from all the answers. In the next step, the client transfers the chosen result to all

the servers. The servers use the result as the initial distance and initial density for the next $i + 1$ iteration. The number of performed iterations and the number of repeated consecutive results are controlled by the client. Operation of the servers is *synchronized* by setting the initial distance and initial density. In this approach client and servers communicate more often, but the obtained results are more precise.

3 Test Results

After the theoretical description of the AEC algorithm we present results of the experiments obtained by means of the described implementations. The main purpose of the experiments was to verify the AEC algorithm correctness and efficiency against various datasets. The AEC algorithm was run with a given set of parameters. The calculated *Eps* parameter was passed to the DBRS algorithm, which was returning the number of created clusters. If the number of clusters declared for a given dataset equaled the number of clusters found by the DBRS, we marked the experiment as a success. If the number of clusters was not equal, we marked the experiment as a failure.

All three implementations (one centralized and two distributed: AO and IT) of the AEC algorithm are written in Java. The experiments were run on machines equipped with Pentium IV 2.8 GHz and 512 MB RAM. The software environment was Windows XP Professional, Java Sun 1.5 and Oracle 10g. The distributed environment consisted of four machines connected with Ethernet 100Mbit network. The communication in distributed implementations was based on Java RMI.

The algorithm was tested on eight various sets of points. The sets were marked from A to H; they vary with cardinality, points distribution and clusters' shapes (fig. 3).

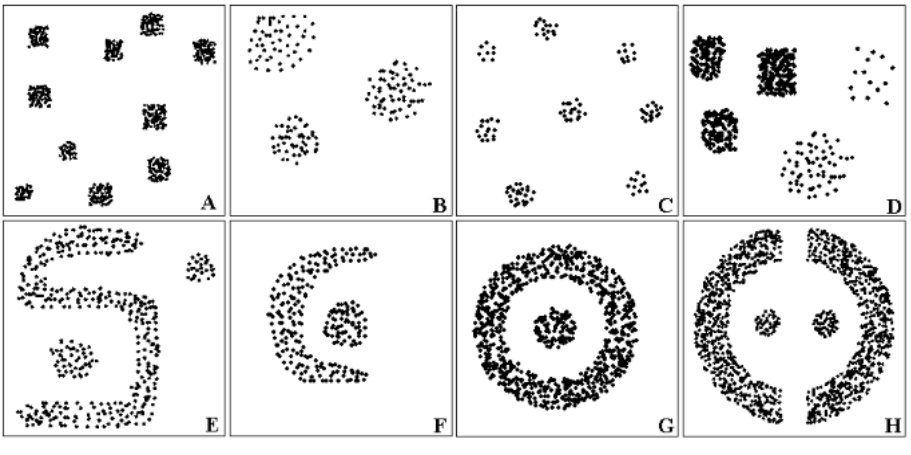

Fig. 3. Sets of points used for testing

The A set contains about 650 points grouped in 10 dense clusters; density of all clusters is very similar. The next set, B, contains about 200 points grouped in three relatively sparse clusters; density of all clusters is similar. The C set contains only about 120 points grouped in eight small clusters. In the D set 400 points are grouped in three dense clusters, one less dense, and one sparse cluster. The E and F sets contain over 400

points. The G and H sets contain respectively 1000 and 1500 points. In all four sets, clusters have similar density but significantly differ in shapes. Small clusters located inside the big ones were intended to disrupt the AEC algorithm when calculating the PIS coefficient. For each dataset we performed a set of experiments with the following parameters:

- the cardinality of the N set was 5, 15, 25 and 35% of the input dataset cardinality,
- the cardinality of the H set was 5, 15, 25 and 35 points for each value of the N set cardinality,
- the number of iterations was set to 10, 20 and 30 for each combination of N and H sets cardinality.

A single test set contained $4 \times 4 \times 3 = 48$ tests. In our tests the iterations were broken if the result of the consecutive iterations was repeated more than 5 times. The iteration breaking was always caused by the number of repeated consecutive results. Thus we can treat the tests for identical cardinality of N and H sets as three repeated tests, which is useful in the presence of the random factor.

The graph in figure 4 illustrates the relative number of investigated points for various sets of points. The number of investigated points calculated as $|N| \cdot |P| \cdot |H|$ was related to the cardinality of the set P, hence we can compare the results for sets of different cardinality in a single plot. In figure 5 we present a graph comparing AEC

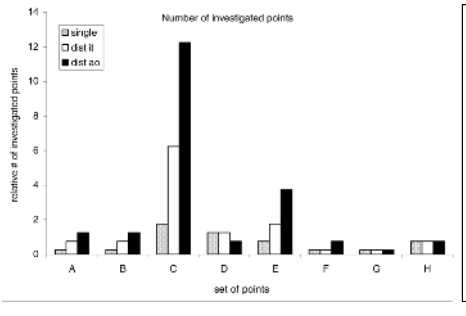

Fig. 4. Relative number of points checked for correct *Eps* calculation

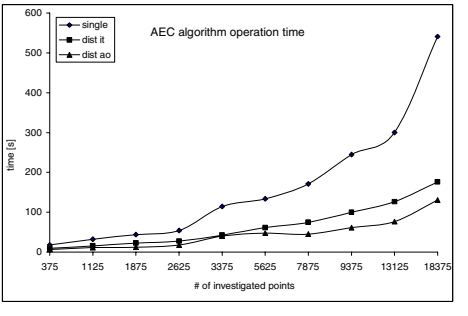

Fig. 5. AEC algorithm operation times as function of investigated points number

algorithm operation times for the three implementation versions. The x axis shows the number of investigated points. The y axis shows AEC algorithm operation times in seconds. We considered only the cases when the algorithm gave the correct results. As expected, the centralized version consumes much more time when compared to the distributed versions. For small cardinalities of investigated points sets (less than 3000) the differences in operation times are not significant. But for greater cardinalities the distributed versions operate much more efficiently. For cardinalities exceeding 10000 points we observe nearly linear speed-up.

Summarizing the tests results we notice that for all tested sets of points the AEC algorithm gives proper results. There are more and less *difficult* sets of points but the algorithm can correctly analyze all of them. The most difficult to analyze are sets of points with a big number of small clusters. The algorithm operation is not disturbed by the differences in densities and/or shapes of the clusters. Also the presence of small clusters inside big ones does not negatively affect the algorithm operation.

The centralized version of the AEC algorithm gives the most accurate results. For all tested sets of points the centralized version always required the smallest N and H sets. This version also needed the smallest number of iterations for obtaining the correct results. The AO distributed version operates most efficiently (is able to examine the biggest number of pairs of points in the shortest time), but on the other hand, the AO version always requires the biggest N and H sets, and the biggest number of iterations. Therefore, the best choice is the iterative distributed version (IT). It is faster than the centralized version and gives better results than the distributed AO version.

4 Conclusions and Future Work

In this paper we addressed the problem of automatic calculation of *Eps* parameter used in density-based clustering algorithms such as DBRS and DBScan. We proposed a solution called AEC algorithm. The algorithm operates iteratively. In every iteration it chooses randomly a fixed number of sets of points and calculates three coefficients: distance between the points, number of points located in a stripe between the points and density of the stripe. Then the algorithm chooses the best possible result, which is the minimal distance between clusters. The calculated result has an influence on the sets of points created in the next iteration.

The AEC algorithm was implemented in one centralized and two distributed versions. We presented test results for a set of eight different sets of points. With appropriately high number of examined points the algorithm was able to calculate the proper *Eps* parameter for all tested sets of points. Our future work includes further improving the AEC algorithm efficiency. We want to optimize the most time-intensive fragment of the algorithm which is calculating the value of the PIS coefficient. We are currently searching for conditions allowing us to skip the PIS coefficient calculation.

References

1. Barclay T., Slutz D.R., Gray J.: TerraServer: A Spatial Data Warehouse, Proc. ACM SIGMOD 2000, pp: 307-318, June 2000
2. http://www.lsgi.polyu.edu.hk/sTAFF/ zl.li/vol_2_2/02_chen.pdf
3. http://maps.google.com
4. Ester M., Kriegel H.-P., Sander J., Wimmer M.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In proc. of 2^{nd} International Conference on Knowledge Discovery and Data Mining, 1996
5. Gorawski, M., Malczok, R.: On Efficient Storing and Processing of Long Aggregate Lists. DaWaK, Copenhagen, Denmark 2005.
6. Papadias D., Kalnis P., Zhang J., Tao Y.: Effcient OLAP Operations in Spatial Data Warehouses. Spinger Verlag, LNCS 2001
7. Wang X., Hamilton H.J.: DBRS: A Density-Based Spatial Clustering Method with Random Sampling. In proceedings of the 7^{th} PAKDD, Seoul, Korea, 2003