A New Fast Clustering Algorithm Based on Reference and Density*

Shuai Ma[1], TengJiao Wang[1], ShiWei Tang[1,2], DongQing Yang[1], and Jun Gao[1]

[1] Department of Computer Science, Peking University, Beijing 100871, China
{mashuai, tjwang, gaojun}@db.pku.edu.cn
http://www.pku.edu.cn

[2] National Laboratory on Machine Perception, Peking University, Beijing 100871, China
{tsw,dqyang}@pku.edu.cn

Abstract. Density-based clustering is a sort of clustering analysis methods, which can discover clusters with arbitrary shape and is insensitive to noise data. The efficiency of data mining algorithms is strongly needed with data becoming larger and larger. In this paper, we present a new fast clustering algorithm called CURD, which means Clustering Using References and Density. Its creativity is capturing the shape and extent of a cluster with references, and then it analyzes the data based on the references. CURD preserves the ability of density based clustering method's good advantages, and it is much efficient because of its nearly linear time complexity, so it can be used in mining very large databases. Both our theoretic analysis and experimental results confirm that CURD can discover clusters with arbitrary shape and is insensitive to noise data; In the meanwhile, its executing efficiency is much higher than R*-tree based DBSCAN algorithm.

1 Introduction and Related Work

Clustering analysis is the process of grouping a set of physical or abstract objects into clusters of similar objects. A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters [1]. Clustering analysis can be used as a stand-alone tool to get insight into data distribution or as a preprocessing step for other data mining algorithms. It is used in many diversified applications such as image compression, market segmentation, and spatial discovery. Clustering is also a challenging field of research where its potential applications pose their own special requirements. The following are typical requirements of clustering in data mining: scalability, ability to deal with different types of attributes, discovery of clusters with arbitrary shape, minimal requirement for domain knowledge

---

to determine input parameters, ability to deal with noisy data, insensitive to the order of input data, high dimensionality, constraint-based clustering, interpretability and usability [1]. A number of different algorithms have been proposed, such as K-MEANS, DBSCAN, BIRCH, CURE, CLIQUE, MAFIA, OPTIGRID, ROCK, CHAMELEON, AMOEBA and C^2P [2-12]. All these algorithms attempt to overcome one or some requirements mentioned above through different approaches, but most of these algorithms cannot reach perfect results. Clustering large and high dimensional data is still an open problem.

K-MEANS is a partitioning clustering algorithm, which takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting intra-cluster similarity is high but the inter-cluster similarity is low. The algorithm attempts to determine k partitions that minimize a certain criterion function. Typically the square-error criterion is used. The time complexity of K-MEANS is O(tkn), where n is the number of objects, k is the number of clusters, and t is the number of iterations. Normally k, t<<n, so it is scalable and efficient in processing large data sets. However K-MEANS have some disadvantages: the users need to specify k, the number of clusters, which is hard to decide in advance; the selection of k initial objects has great effect on the final clustering result and algorithm's efficiency; the partition based clustering algorithms could split large clusters to minimize the square-error [5].

CURE is a bottom-up hierarchical clustering algorithm, which starts by placing each object in its own cluster and then merges similar atomic clusters into larger and larger clusters, until the number of clusters is k. At each step, the pair of clusters merged are the ones between which the distance is the minimum. CURE points out that neither the centroid-based approach nor the all points approach works well for non-spherical or arbitrary shaped clusters, so CURE adopts a middle-ground between the centroid and the all points extremes. CURE chooses a constant number c of well-scattered points in a cluster, which capture the shape and extent of the clusters, thus the ability of processing arbitrary clusters is improved. The time complexity of CURE is O(n^2) for low dimensional data, and O($n^2 logn$) for high dimensional data, so CURE adopts random sampling and partitioning technologies in order to process large data sets.

DBSCAN is a density based clustering algorithm, which grows regions with sufficiently high density into clusters and defines a cluster as a maximal set of density-connected points. DBSCAN discovers clusters with arbitrary shape with noise and is insensitive to the order of input data. The time complexity of DBSCAN is O(n^2); with the support of spatial access methods such as R*-tree its time complexity reduces to O($nlogn$). We should point out that DBSCAN does not consider the time of establishing R*-tree, which often consumes a lot of time.

CLIQUE is a grid and density based clustering algorithm, which has the fastness of grid-based approaches and the ability to process high dimension data. CLIQUE considers little about the distribution of data when partitioning data into grid cells and uses the statistical information stored in the grid cells, so the clustering quality is lowered.

In this paper we propose a new fast clustering algorithm named CURD (Clustering Using Reference and Density). The remainder of the paper is organized as follows. In

section 2, we present CURD algorithm. In section 3, we analyze the performance from different points of view. Discussion remarks and future work are made in section 4.

2   CURD Algorithm

CURD algorithm is first motivated by CURE algorithm, and it uses references to correctly represent the shape and extent of a cluster. References are not real input points, but virtual points, which are different from the represents of CURE. In addition, the number of references used to represent a cluster is not constant, which is more reasonable than using fixed number as CURE. CURD algorithm takes the density approach, similar to DBSCAN, to eliminate the effect of noise data. The time complexity of CURD is approximately equal to K-MEANS's, so its efficiency is very high. The references of CURD consider much about the spatial geometric feature of data, thus the clustering quality is higher than grid-based clustering algorithms. High dimensional data can be transformed into one single dimension space based on distance [13,14], so CURD can process high dimensional data from this point. The distance computation of high dimensional data is the same as 2 dimensional data, so we use Euclidean distance and 2 dimensional spaces to analyze CURD algorithm.

2.1   Definitions

Definition 1: (density of point) Given a point p and a distance dRadius, the number of points in the circular region, where p is the center and dRadius is the radius, is called the density of p based on distance dRadius, denoted by Density(p,dRadius).

Definition 2: (reference) Given a point p, a distance dRadius and a threshold t, if Density(p,dRadius)≥t, p is called a reference, and t is called the density threshold.

References are not real points in input data, but virtual points.

Definition 3: (representing region) Each reference p represents a circular region, where p is the center and dRadius is the radius. The region is called p's representing region.

Definition 4: (neighboring references) Points p and q are references based on distance dRadius and density threshold t, if Dist(p,q)≤2dRadius, which represents the distance between p and q is equal to or less than 2 times of dRadius, p and q are called neighboring references.

In fact if the representing circular regions of the two references are tangent, intersectant or equal to each other, the two references are neighboring references.

2.2   Clustering Algorithm

Now we describe the details of our clustering algorithm. CURD first finds the references which can correctly represents the shape and extent of input data; then it establishes the mapping between points and corresponding references; the references are then classified, and the references in the same class contains the basic information of a

cluster; points are mapped to the corresponding clusters at last. Fig. 1. gives an over-view of CURD algorithm.
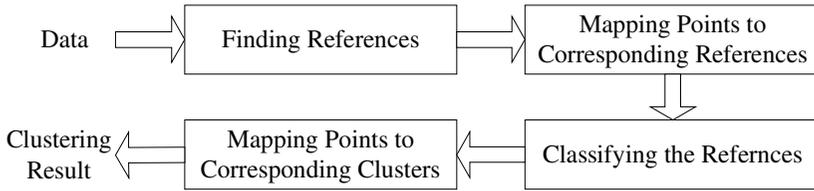
Fig. 1. Overview of CURD

2.2.1 Data Structure

For the importance of references, we first describe the data structure of references. Each reference record contains four kinds of data: X_m and Y_m, which is the reference's X coordinate and Y coordinate; X_s and Y_s, which is the sum of X coordinate and Y coordinate of all the points within the reference's representing region; N_s, which is the density of the reference or the number of points in the reference's representing region; the point set P_s, which is the set of all the points within the reference's representing region.

2.2.2 Finding References Procedure

Finding references procedure is divided into two steps: candidate references are found in the first step; the candidate references whose density is lower than the density threshold are filtered in the second step, and the remaining candidate references are real references. Filtering candidate references is very simple, so we mainly describe the procedure of finding candidate references (Prog. 1.). The Prog. 1. shows us that Finding_Candidate_References procedure iteratively calls Single_Scan procedure and Regenerate_Candidate_References procedure, thus we will describe them firstly.

Prog. 1. Program of Finding_Candidate_References Procedure

```
Finding_Candidate_References(PointSet,dRadius,Iterate)
 {Input:data set:PointSet,distance:dRadius,number of
 iterative times:Iterate;
 Output:candidate reference set:CandidateReferenceSet};
 begin
   CandidateReferenceSet := ∅; I := 0;
   repeat
     I :=I + 1;
     Single_Scan(PointSet,dRadius,CandidateReference-
     Set);
     Regenerate_Candidate_References(CandidateRefere-
     nceSet);
   until I = Iterate;
   Single_Scan(PointSet,dRadius,CandidateReferenceSet);
 end.
```

Each call of Single_Scan procedure produces new candidate reference set, and its main operation is adding point p's information to the corresponding candidate reference. If the distances between p and all candidate references are larger than the distance dRadius, a new candidate reference is generated and added into the candidate references set; otherwise p's information is added to all the candidate references, whose distances with p are either equal to or less than the distance dRadius (Prog. 2.).

Prog. 2. Program of Single_Scan Procedure

```
Single_Scan(PointSet,dRadius,CandidateReferenceSet)
 {Input: data set: PointSet, distance: dRadius,
 candidate reference set: CandidateReferenceSet;
 OutPut:candidate reference set:CandidateReferenceSet};
 begin
   I := 0;
   repeat
     I := I + 1;
     For each candidate referenct R, whose distance
     with Pi is equal to or less than dRadius
     {Adding Pi's information to every candidate re-
     ference R}
       R.Xs := R.Xs + Pi.X;
       R.Ys := R.Ys + Pi.Y; R.Ns := R.Ns + 1;
     If the distances between Pi and all candidate
     references are larger than dRadius then
       begin
         Generated a new candidate reference R;
         R.Xm := Pi.X; R.Ym := Pi.Y;
         R.Xs := Pi.X; R.Ys := Pi.Y; R.Ns := 1;
         CandidateReferenceSet := CandidateReference-
         Set + {R};
       end;
   until I = PointSet.Size;
   Single_Scan(PointSet,dRadius,CandidateReferenceSet);
 end.
```

Each call of Regenerate_Candidate_References procedure produces new candidate references. The mean of all the points in the representing region of a candidate reference replaces the candidate reference itself. Since the sum of X coordinate and Y coordinate of all the points in the references' representing region and the number of points in the references' representing region have been computed and stored in the Single_Scan procedure, the computation of the new candidate reference R´ of each candidate R is very simple, where $R´.X_m = R.X_s/R.N_s$, $R´.Y_m = R.Y_s/R.N_s$, $R´.X_s = 0$, $R´.Y_s = 0$, $R´.N_s = 0$.

Finding_Candidate_References procedure iteratively calls Single_Scan procedure and Regenerate_ Candidate_References procedure, and each call of them is prone to generate candidate references that can better represent the shape and extent of input data. Through a series of optimization the final candidate references can coarsely represent the spatial geometric feature of input data. The candidate references are then

filtered, and the candidate references whose densities are lower than the density threshold are removed. By this way the effect of noise data can be effectively eliminated, thus the references can quite correctly represent the spatial geometric feature of input data.

2.2.3 Mapping Points to Corresponging References

Lemma 1. In CURD, mapping point p to which reference has no effect on the final clustering result, only if the distance between p and the reference is equal to or less than dRadius.

Proof: Suppose there exist two references R1 and R2, and the distances between p and them are either equal to or less than dRadius. From the famous theorem of triangles: one side's length is less than the sum of the other two sides' length, it is easy to know that the distance between R1 and R2 is less than 2 times of dRadius, so R1 and R2 are neighboring references (left part of Fig. 2.). In high dimensional space, any three points that are not in the same line form a plane, so the theorem of triangles still works. In the special case, point p, references R1 and R2 are in the same line (right part of Fig. 2.), it is easy to known that the distance between R1 and R2 is equal to or less than two times of dRadius, so R1 and R2 are still neighboring references. In CURD algorithm, neighboring references contain the basic information of a cluster, and data belonging to the neighboring references are in the same cluster, thus either mapping p to R1 or R2, p belongs to the same cluster.
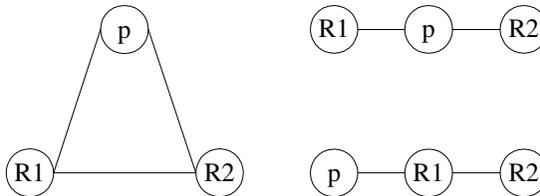
Fig. 2. Relative positions of p and R1, R2

Mapping_References_and_Points procedure establishes the mapping between references and the points within their corresponding representing regions. Theoretically, mapping point p to the reference that is closest to it is more reasonable. From Lemma 1, we know that mapping point p to which reference has no effect on the final clustering result, only if the distance between p and the reference is equal to or less than dRadius. Mapping_References_and_Points procedure orderly computes the distances between point p and the references, and if there exists a reference whose distance with p is equal to or less than dRadius, the mapping between p and the reference is established. After that, Mapping_References_and_Points procedure continues to process the next point. By that way, the algorithm's efficiency is improved (Prog. 3.).

Prog. 3. Program of Mapping_References_and_Points Procedure

```
Mapping_References_and_Points(PointSet,ReferenceSet,t,
dRadius)
```

```
{Input: data set: PointSet, reference set: Reference-
Set, density threshold: t, distance: dRadius;
OutPut: reference set: ReferenceSet};
begin
  For each reference R in ReferenceSet
  R.Ps := ∅;
  I := 0;
  repeat
    I :=I + 1;
    If the distance between Pi and some reference R
    is equal to or less than dRadius then
      R.Ps := R.Ps + {Pi};
    else if the distances between Pi and all the ref-
    erences are larger than dRadius then
      Pi is considered as noise and is discarded
  until I = PointSet.Size;
end.
```

2.2.4 Classifying the References

Given a reference set based on distance dRadius and threshold t, and if the distance between reference R_1 and R_2 is equal to or less than 2 times of dRadius, R_1 and R_2 are neighboring references. We can describe the reference set with an undirected graph, where the reference is vertex, and any two neighboring references form an edge. The references that are in the same connected sub-graph form a class.
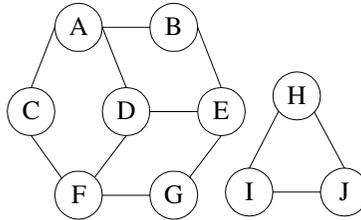
Fig. 3. Graph G

For example, {A, B, C, D, E, F, G} are the vertexes in the same connected sub-graph of graph G, so they form a class C_1={A, B, C, D, E, F, G}; {H, I, J} are also the vertexes in the same connected sub graph, so they form another class C_2={H, I, J} (Fig. 3.). It is easy to find the vertexes that are in the same connected subgraph through the graph's broad first search algorithm (BFS), so we do not describe the details of classifying the references. The references in a class contain the basic infor-mation of a cluster, and they correctly represent the spatial geometric feature of the clusters.

2.2.5 Mapping Points to Corresponding Clusters

The mapping $f_{points2references}$ between the points and the references has been established, and the procedure of classifying the references has established the mapping $f_{references2classes}$ between the references and the classes. The data belonging to the references, which

are in the same class, form a cluster, so the mapping $f_{points2clusters}$ between the points and the clusters is very simple: $f_{points2clusters=}f_{points2references}f_{references2classes.}$

2.2.6   Time and Space Complexity

Suppose the number of input data is n, the maximum number of candidate references in the procedure of finding candidate references is k, the iterative number is i, the number of references is m and the number of clusters is c.

It is easy to know that the time complexity of Single_Scan procedure is O(kn) and the complexity of Regenerate_Candidate_References procedure is O(k), so the time complexity of Finding_Candidate _References procedure is O($ikn+(i-1)k$); Filtering candidate references need O(k) time. Thus the time complexity of finding references procedure is O($ikn+(i-1)k$)+O(k). Every point could find its corresponding references in time O(m), so mapping points to corresponging references procedure needs O(mn) time. The classification of references needs O(m^2) time using graph's breadth first search algorithm (BFS). In fact the mapping between data and corresponding clusters has been established after the classification of references. From the above analysis, we know the time complexity of CURD algorithm is O($ikn+(i-1)k$)+O(k)+O(mn)+O(m^2), Normally $k,i,m<<n$, so the time complexity approximately equals to O($ikn+mn$), which has almost the same time complexity as K-MEANS.

The space complexity of CURD is O(n)+O(k)+O(c).

3   Performance Analysis

In this section, we evaluate the performance of CURD. We experimented three data sets containing points in 2 dimensions (Fig. 4.). DS1 comes from the Data set 1 of CURE; DS2 comes from database 3 of DBSCAN; we use the DBSCAN program that Dr. Jörg Sander provides to generate DS3, which is similar to the database 2 of DBSCAN. The data sizes are respectively 100000, 203 and 306.

All experiments are run on IBM Netfinity 5500: two X86 Family 6 Model 10 Stepping 1 GenuineIntel~700Mhz CPU, 512M main memory and 20G hard disk; the operation system is Microsoft Windows 2000 Server.
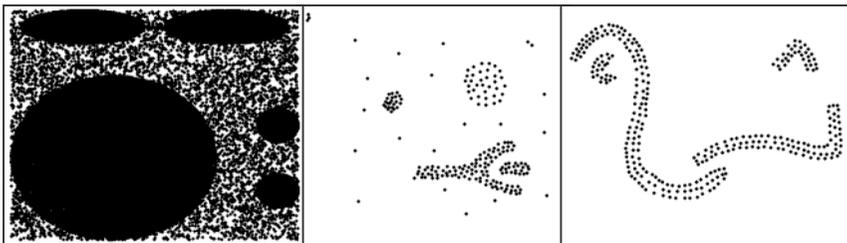
Fig. 4. Data sets

3.1   Comparison of Clustering Quality

Both CURD and DBSCAN are density based clustering algorithms; in the meanwhile
the input parameters dRadius and t of CURD are similar to the input parameters of
Eps and MinPts of DBSCAN, so we choose DBSCAN to compare the clustering qual-
ity with CURD.
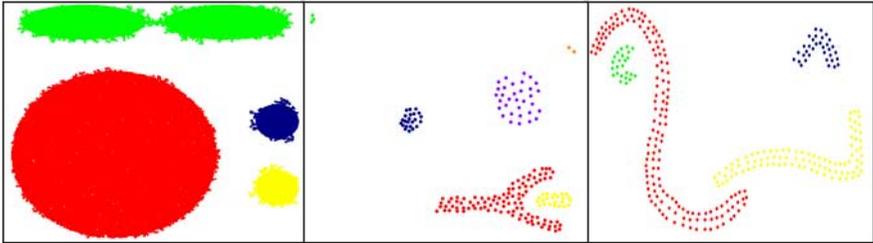
Fig. 5. Clustering results of DBSCAN, with the corresponding parameter Eps and MinPts are
(0.5,6), (2,2), and (2,2) respectively
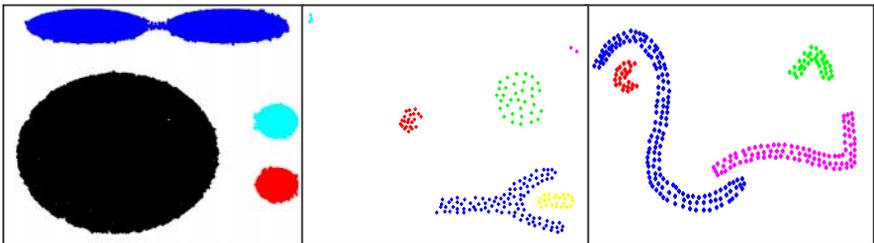
Fig. 6. Clustering results of CURD, with the corresponding parameter Radius and t are (0.5,60),
(2,2), and (2,2) respectively

Fig. 6. shows the clustering results of CURD, from which we can see that CURD
can effectively process clusters with arbitrary shape and is insensitive to noise data.
Fig. 5. shows the clustering results of DBSCAN, from which we can see that the clus-
tering results of DBSCAN and CURD are very similar to each other.
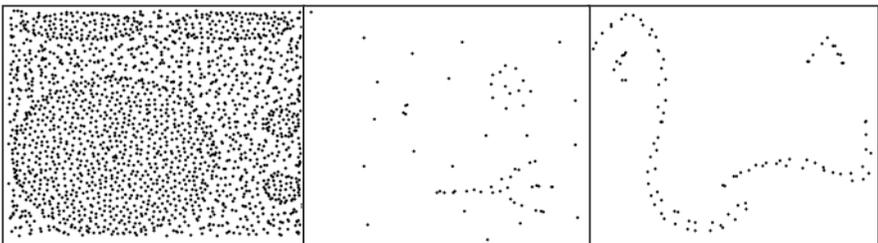
Fig. 7. Candidate references

3.2   Candidate References and References

Fig. 7. and Fig. 8. show the candidate references and references of DS1, DS2 and DS3. Fig. 7. shows that candidate references coarsely represent the spatial geometric feature of data set, but it is affected by the noise. In Fig. 8., the references correctly represent the spatial geometric feature of data set by filtering interferential candidate references whose density is lower than the density threshold.
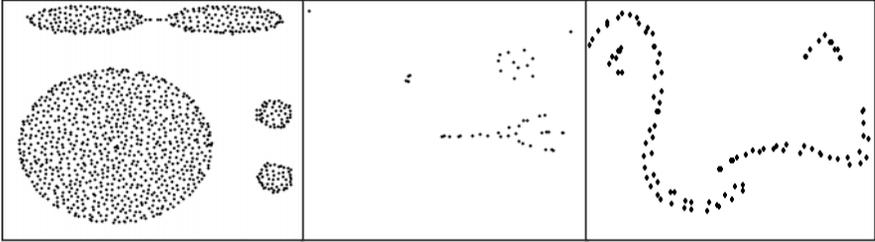
Fig. 8. References

3.3   Sensitive to Parameters

CURD has three main parameters: distance dRadius, density threshold t, and iterative times of Iterate. The values of the three parameters have effects on the clustering result and executing efficiency of CURD. In our experiments, we found that the CURD's clustering results are similar to DBSCAN's if the values of dRadius and t are the same as the values of Eps and MinPts respectively. In DBSCAN it is hard to determine the values of Eps and MinPts, but in CURD, we can set the values of dRadius and t easily. CURD generates better results if the value of dRadius equals to 1/50~1/100 of the whole data space. For example, DS1 is a 30X30 two dimensional data space, so the value of dRadius is set to 0.5; DS2 and DS3 are 100X100 two dimensional data space, so the value of dRadius is set to 2. The density threshold t can be set to the mean value of all candidate references' density, which can be computed automatically. As to the parameter Iterate, we should point out that all our experimental results are generated with the value of Iterate equal to 4. The bigger the values of Iterate, the better the clustering results, but longer the executing time.

3.4   Comparison of Executing Time with DBSCAN

There are mainly three reasons why we choose traditional R*-tree based DBSCAN algorithm to compare with CURD, which are as follows:

The two algorithms are both density based clustering algorithms;

Their key input parameters are similar to each other;

The time complexity of R*-tree based DBSCAN algorithm is O($nlogn$), so it is already a very fast clustering algorithms.

Fig. 9. shows that the CURD's executing efficiency is much higher than DBSCAN's, so CURD is a very efficient clustering algorithm. In addition, R*-tree

based DBSCAN algorithm need the support of R*-tree index, which needs much time to be established, so if the time of establishing index is considered, CURD is much more efficient than DBSCAN.

Fig. 10. shows that the executing details of CURD. Finding reference and mapping points to the corresponding references take most part of the executing time, especially finding references. Once mapping points to their corresponding references has been finished, CURD only takes little time to process the rest work.
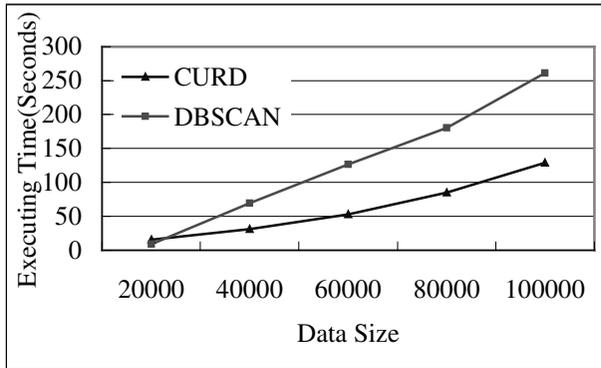
Fig. 9. Comparison of executing time
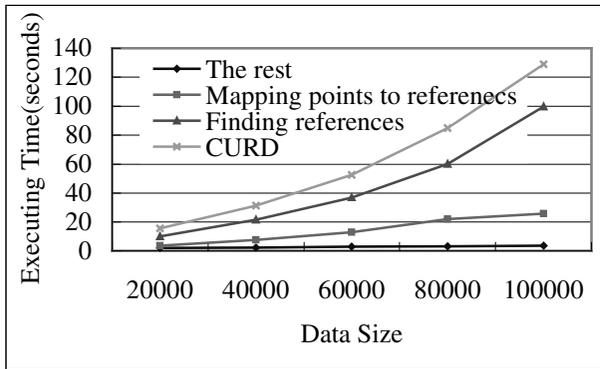
Fig. 10. Executing time details

4   Discussion and Future Work

In this paper, we propose a new method (using references) to represent the spatial geometric feature of data space, and we present a new fast clustering algorithm based on references and density. CURD not only preserves the advantages of density based clustering algorithms, which can discover clusters with arbitrary shape and is insensitive to noise data, but also it can reach high efficiency because of its approximately linear time complexity, which is nearly the same as K-MEANS'. Both our theoretic analysis and experimental results confirm the above conclusions.

High dimensional data can be transformed into one single dimension space based on distance [13,14], so CURD can process high dimensional data from this point. One of our future work is to experiment its performance in high dimensional space.

Last but not least, our algorithm finds references that can correctly represent the spatial geometric feature of data space, so we are thinking of sampling methods using this technology.

References

1. Jiawei Han, Micheline Kambr. Data mining concepts and techniques, Morgan Kaufmann Publisher (2000) 145–176.
2. Anderberg, M. R. Cluster analysis for applications, Academic Press (1973).
3. M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density based algorithm for discovering clusters in large spatial databases with noise. Proceedings of International Conference on Knowledge Discovery and Data Mining (1996).
4. T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. Proceedings of ACM SIGMOD International Conference on Management of Data, Montreal, Canada (1996) 103–114.
5. S. Guha, R. Rastogi and K. Shim. CURE: An efficient clustering algorithm for large databases. Proceedings of ACM SIGMOD International Conference on Management of Data, New York (1998) 73–84.
6. R. Aggrawal, J. Gehrke, D. Gunopulos, P. Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications, Proceedings of ACM SIGMOD International Conference on Management of Data, Seattle, Washington (1998) 94–105.
7. Goil, Saniay, Harasha Nagesh and Alok Choundhary. MAFIA: Efficient and scalable Subspace Clustering for Very Large Data Sets. Technical Report Number CPDC-TR-9906-019, Center for Parallel and Distributed Computing, Northwestern University (1999).
8. Hinneburg, Alexander and Daniel A.Keim. Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. Proceedings of the 25th VLDB Conference, Edinburgh, Scotland (1999).
9. Guha, S., Rastogi. R. and Shim K. Rock: A Robust Clustering Algorithm for Categorical Attributes, Proceedings of the International Conference on Data Engineering, Sydney, Australia (1999) 512–521.
10. Karypis George, Eui-Hong Han, and Vipin Kumar. CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling. IEEE Computer (1999) 68–75.
11. Estivill-Castro,Vladimir and Ickjai Lee. AMOEBA: Hierarchical Clustering Based on Spatial Proximity Using Delaunay Diagram. Proceedings of the 9th International Symposium on Spatial Data Handling. Beijing, China (2000).
12. Alexandros Nanopoulos, Yannis Theodoridis,Yannis Manolopoulos. C2P: Clustering based on Closest Pairs. Proceedings of the 27th VLDB Conference, Roma, Italy (2001).
13. S. Berchtold, C. Bohm, and H-P. Kriegel. The pyramid-technique: Towards breaking the curse of dimensionality. Proceedings of ACM SIGMOD International Conference on Management of Data (1998) 142–53.
14. C. Yu, B. C. Ooi, K.-L. Tan, and H. V. Jagadish. Indexing the Distance: An Efficient Method to KNN Processing. Proceedings of 27th VLDB Conference, Roma, Italy (2001).