GraphZip: A Fast and Automatic Compression Method for Spatial Data Clustering

Yu Qian Kang Zhang
Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75083-0688, USA
{yxq012100, kzhang}@utdallas.edu

ABSTRACT

Spatial data mining presents new challenges due to the large size and the high dimensionality of spatial data. A common approach to such challenges is to perform some form of compression on the initial databases and then process the compressed data. This paper presents a novel spatial data compression method, called GraphZip, to produce a compact representation of the original data set. GraphZip has two advantages: first, the spatial pattern of the original data set is preserved in the compressed data. Second, arbitrarily dimensional data can be processed efficiently and automatically. Applying GraphZip to huge databases can enhance both the effectiveness and the efficiency of spatial data clustering. On one hand, performing a clustering algorithm on the compressed data set requires less running time while the pattern can still be discovered. On the other hand, the complexity of clustering is dramatically reduced. A general hierarchical clustering method using GraphZip is proposed in this paper. The experimental studies on four benchmark spatial data sets produce very encouraging results.

Keywords:

Spatial databases, data compression, clustering

1. INTRODUCTION

As the size of spatial data increases exponentially and the structure of data becomes more complex, data mining and knowledge discovery techniques become essential tools for successful analysis of large spatial data sets [12]. Spatial clustering, which groups similar spatial objects into classes, is an important component of spatial data mining [8]. There have been many contributions in spatial clustering during recent years [16, 4, 6, 7, 11, 10]. However, most approaches have focused on the cluster quality. Few existing algorithms perform efficiently when the data set to be clustered is large on both the size and the dimensionality [13]. The size of the data being clustered plays an important role in the running time of a clustering algorithm. Most spatial clustering algorithms define the similarity between two data points as a certain kind of distance between them. Thus clustering without preprocessing requires calculating the distance between all pairs of data points, an $O(n^2)$ operation. High dimensionality further complicates the scalability issue. Many graph-based clustering approaches model data sets using *k*-nearest/mutual graphs while the construction of such graphs requires $O(n^2)$ time for high dimensional data when $k>1$.

This paper proposes a novel spatial data compression method that can handle high dimensional data automatically and efficiently. The key idea involves iteratively constructing *1*-neareset neighbor graphs on the given data set to merge the closest data together until the size of the data set drops significantly. Then clustering is performed on the compressed data. Because *1*-nearest neighbor graphs can be constructed efficiently even for high dimensional data and the size of the compressed data is much smaller than that of the original data set, the whole clustering process becomes very efficient. Our analysis will show that performing a hierarchical combination on the output of GraphZip can be completed in linear time while traditional hierarchical method requires $O(n^2)$ time.

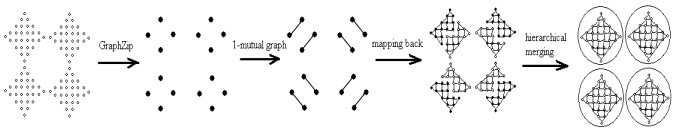


Figure 1. The four steps of the proposed clustering process

As illustrated in Fig. 1, there are four steps in the proposed clustering process: Step 1 is GraphZip, which transforms the original data set into a compressed one. Each data point of the compressed data set represents a group of data points of the original data set; Step 2 creates groups in the compressed data set with *1*-mutual neighborhood graph; Step 3 maps back the grouping information of the compressed data set to the original data set to obtain corresponding sub-graphs before hierarchical merging; Step 4 merges the sub-graphs hierarchically according to a proposed combination criterion. The whole process requires $O(nlogn)$ time.

The contributions of this paper can be summarized as follows:

1) A compact representation of a spatial data set, which contains the same pattern of the original data set with significantly fewer points, and the corresponding transformation method between the compact representation and the original data set, called GraphZip. Such a compact representation can be used as inputs to existing clustering approaches to improve the clustering efficiency.

2) A method to break bridges between natural clusters. By creating a *1*-mutual graph for the given spatial data set, each obtained connected component of the graph contains no bridge and is part of either a natural cluster or outliers.

3) A hybrid hierarchical merging criterion that is robust to noise and combines the sub-graphs correctly. Working on the result of GraphZip, the hierarchical combination can be completed in $O(n)$ time while traditional hierarchical approaches require $O(n^2)$ time.

The rest of this paper is organized as follows. Section 2 reviews related work. The details of GraphZip and the bridge-breaking process are described in Section 3. Section 4 presents a hierarchical clustering method integrated with GraphZip. The experimental results on four benchmark spatial data sets are reported in Section 5. Section 6 concludes the paper.

2. RELATED WORK

There has been extensive literature on clustering algorithms, including *K*-means [14], CLARANS [16], BIRCH [22], DBSCAN [4], CURE [6], ROCK [7], CHAMELEON [11], Canopy [13], AUTOCLUST [5], RandomWalk [10], and SNN [3]. From the perspective of whether they use preprocessing methods or not, the

SAC '04, March 14-17, 2004, Nicosia, Cyprus

clustering algorithms can be classified into two categories: most approaches cluster the original data set without any compression or preprocessing while several other approaches transform the given data set before the real clustering begins. The reasons that most clustering approaches do not use compression are: 1) there are not many available appropriate compression methods. 2) Compression is usually lossy. As a result, the quality of the resulting clusters will drop accordingly. As noted by Han *et al.* [9], the challenge is to compress data in such a way that the speed of clustering increases substantially with minimal compromise in cluster quality.

There are three representative structures for approaches using data compression or preprocessing: KD-tree [1, 15], CF tree [22], and Canopy [13].

KD-tree was first proposed by Bentley [1]. Moore [15] provides a multiresolution version for efficient EM-style clustering of many elements, but requires that the dimensionality of each element be small, as splits are generally made on a single attribute. Construction of a KD-tree recursively partitions the data into subgroups and provides more opportunity for a fast nearest neighbor search in the given data set. Almost all of the KD-tree-based methods suffer from doing hard partitions, where each item must be on one side of each partition. If an item is put on the wrong side of a partition there is no way to correct the error.

Zhang *et al.* propose two important concepts in BIRCH [22]: *CF (Clustering Feature)* and *CF* tree. *CF* can be regarded as summary information about a cluster. *CF* tree is a height-balanced tree that can be built dynamically as new data objects are inserted. If the memory cannot hold the *CF* tree after a new data object is inserted, some nodes of *CF* tree can be merged into a new one, which rebuilds the *CF* tree. In order to control the rebuilding, BIRCH introduces two thresholds B and T to limit the number of branches of a *CF* tree and the *diameter* of each cluster. Several concepts like *radius*, *diameter*, and *centroid* are used in BIRCH to describe the distance properties of a cluster, which leads to a possible drawback [19], i.e., it may not work well when clusters are not "spherical".

The key idea of Canopy [13] involves using a fast and approximate distance measure to efficiently divide the data into overlapping subsets called *canopies*. Then clustering is performed by measuring exact distances only between points that occur in a common canopy. Experiments show that Canopy can reduce computation time over a traditional clustering approach by more than an order of magnitude. Canopies are created with the intention that points not appearing in any common canopy are far enough apart that they could not possibly be in the same cluster. However, since the distance measure used to create canopies is approximate, this intention is not always guaranteed.

GraphZip differs from the aforementioned approaches in three ways. First, none of the three structures preserves the spatial pattern of the original data in the compact representation. Second, GraphZip is automatic and requires no threshold to control the size or shape of the clusters. Thus the discovery of natural clusters will not be hindered by inappropriate parameters. Third, GraphZip is not sensitive to the input order of the data since it uses a graph-based approach. Different input orders of data points produce the same graph.

Among the clustering methods without compression or preprocessing, CHAMELEON [11] is a well-known representative. CHAMELEON tries to improve the clustering quality by using an elaborate criterion when merging two clusters. It proposes a more objective definition of similarity, which is composed of relative inter-connectivity and relative closeness. Two clusters will be merged if the inter- connectivity and closeness of the merged cluster is very similar to the inter-connectivity and closeness of the two individual clusters before merging. CHAMELEON needs several parameters to perform satisfactory clustering, and requires $O(nm+n\log n+m^2\log m)$ time with n number of points (m is the number of initial clusters that is about $0.03n$), not including the time for a k-nearest graph construction. Section 5 will show that our approach can obtain comparable results with those of CHAMELEON in a much faster speed yet without using any parameters.

3. EFFICIENT DATA COMPRESSION WITH GRAPHZIP

The first phase of GraphZip is to construct a *1*-nearest neighbor graph. As noted by Karypis et al. [11], the advantages of representing data using k-nearest or k-mutual graph include: firstly, data points that are far apart are completely disconnected from the graph. Secondly, the constructed graph is able to represent the natural density dynamically. In a dense region, the neighborhood radius of a data point is smaller than that of a data point in a sparse region. Thirdly, the number of graph edges is linear to the number of vertices.

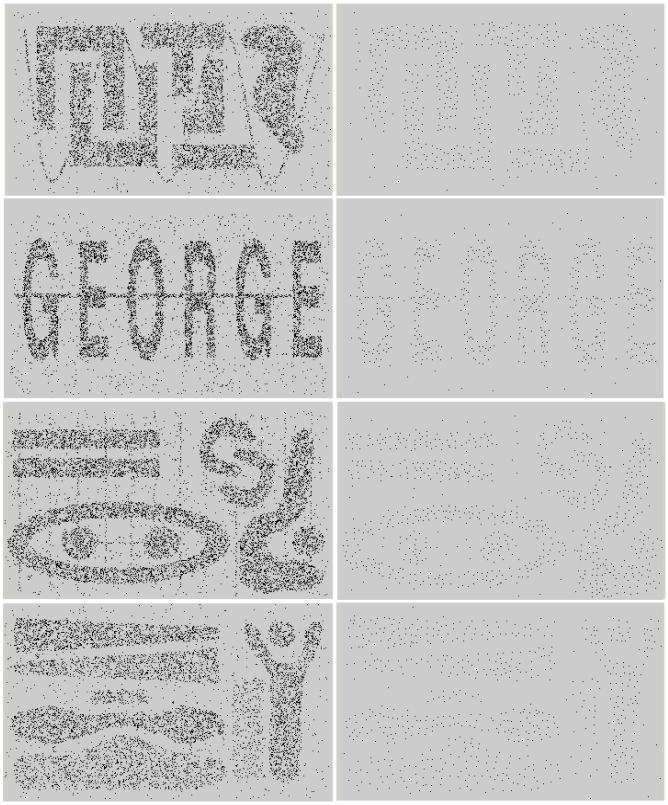
Algorithm *GraphZip (Data Set* $\mathbb{D}$*)*
begin
  Construct 1-nearest neighbor graph G for $\mathbb{D}$*;*
  Create an empty data set $\mathbb{D}'$*;*
  For each connected-component C of G:
    Generate a point p that is located at the center of C;
    Add p to $\mathbb{D}'$ *and update the mapping file;*
  if $O(|\mathbb{D}'|) \le O(\sqrt{n})$ **return** $\mathbb{D}'$*;*
  else *GraphZip(*$\mathbb{D}'$*);*
end

Figure 2. The *GraphZip* Algorithm

As shown in Fig. 2., GraphZip is an iterative process, which accepts the output of the last running cycle as the input and decreases the size of the input data set iteratively. In each running cycle, a *1*-nearest neighbor graph is constructed. Then a new point is placed at the center of each connected component of the constructed graph. All the new points form a new data set, which is the output of this running cycle and will be the input of the next cycle. The mapping information between the new point and the set of points in the connected component is recorded in a mapping file, which is updated in each running cycle. The process is repeated until the size of the output data set reaches a predefined level. When the number of points, n, in the original data set, is decreased to $O(\sqrt{n})$, the iteration stops. The reason we choose $\sqrt{n}$ as the stopping threshold is because it can be valid for most spatial databases. Observation on many spatial databases reveals a fact that to keep the pattern of a spatial data set requires only a small portion of the original data points. In our experiments $O(\sqrt{n})$ can be always reached without losing spatial patterns.

GraphZip aims at improving both efficiency and effectiveness. From our experiments on many data sets, we found that the complexity of group merging decreases greatly when the number of initial groups is relatively small. In other words, if we can simplify the input graph while maintaining its pattern, both efficiency and effectiveness of hierarchical combination will be dramatically improved. GraphZip is designed with three characteristics: a) maintaining the original graph pattern with a more uniform distribution of the data points inside each cluster, b) reducing the size of the data set and the number of the sub-graphs used in later steps, and c) the distance between the data points inside a cluster increasing no faster than the distance

between the data points between clusters. The effect of this last characteristic is that bridges between natural clusters are stretched and finally broken by a later step.



(a) (b)

Figure 3. The data set (a) before (b) after applying GraphZip.

Fig. 3 shows the experimental results of applying GraphZip to four benchmark spatial data sets used in CHAMELEON [11]: the number of data points is greatly reduced while the spatial pattern is preserved. In Fig. 3 (b), the points in the natural clusters have a more uniform distribution compared with the original data in Fig. 3(a), the number of points has been dramatically reduced, and no natural clusters are mixed. This intuitively demonstrates the first characteristic mentioned above. We argue that this result is general enough to be applicable to any graph-theoretical clustering algorithms for improving efficiency.

The time complexity of GraphZip depends on the time to construct *1*-nearest neighbor graph and finding the connected components. The construction of *1*-nearest neighbor graph equals to the *all-nearest-neighbors problem*, i.e., given a set S of n points in $\mathbb{R}^d$, we want to compute for each point p of S another point of S that is closest to p. The first $O(n\log n)$-time algorithm for the all-nearest-neighbors problem for an arbitrary dimension d was given by Clarkson [2], using randomization. Vaidya [18] solves the problem deterministically, in $O(n\log n)$ time. Vaidya's algorithm can be implemented in the algebraic computation tree model and is therefore, optimal [17].

To find the connected components of an undirected graph with n vertices and m edges requires $O(n+m)$ when using a *DFS* or *BFS* search tree. Since $m \le kn$ in k-nearest neighbor graph while $k=1$ in GraphZip, i.e., $m \le n$, the time complexity of the first iteration of GraphZip is $O(n\log n+2n)=O(n\log n)$. Now let us analyze how many iterations GraphZip needs to run before reaching the graph size of $O(\sqrt{n})$ points.

Theorem 3.1 GraphZip requires $O(\log(\sqrt{n}))$ iterations to compress the size of a given data set from n to $\sqrt{n}$.

Proof. Let us suppose that after the first iteration, there are X_1 connected components in the *1*-nearest graph of the given data set. According to the definition of k-nearest neighbor graph, every node has at least k edges connected and $k \ge 1$, so there is no isolated node. In other words, each connected-component in the graph has at least two nodes, i.e., $X_1 \le n/2$. Generally, let X_i denote the number of connected components after i iterations and $X_o=n$, we have $X_i \le X_{i-1}/2$. Solving the formula $X_t=\sqrt{n}$, i.e., $n(1/2)^t=\sqrt{n}$ based on the recursive expression for variable t leads to $t \le \log(\sqrt{n})$. □

Theorem 3.1 indicates that the total time complexity of GraphZip is the sum of the $\log(\sqrt{n})$ iteration steps: $O(n\log n+(n/2)\log(n/2)+\ldots+2\sqrt{n}\log(2\sqrt{n}))<O(2(n\log n+2n))=O(n\log n)$. For example, a given graph contains a million of vertices can be compressed into 1000 nodes in at most 10 iterations. In real experiments every connected component usually contains more than *2* nodes, so the running time is much less than expected. In our experiments the program iterations are approximately $\log_4(\sqrt{n})$ because every connected component contains *4* nodes on average, as shown in Fig. 4. Theorem 3.1 concludes the second characteristic of size issue of GraphZip.

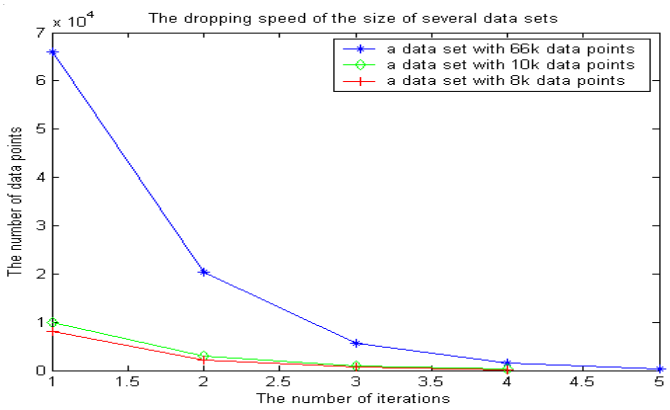


Figure 4. The dropping speed of the size of several testing data sets when applying GraphZip

After applying GraphZip, the next task is to prepare the initial groups for the later hierarchical process. The construction of initial groups is composed of two steps: the first step constructs a *1*-mutual neighbor graph of the zipped data set, the second step maps back each connected component of the *1*-mutual graph to the original data set as the initial groups. Constructing a *1*-mutual graph on the zipped data set can further reduce the number of initial groups without producing any bridges. Recall the mapping file used in GraphZip, each data point in the zipped data set can be mapped to a set of original data points. If two vertices of the *1*-mutual graph of the zipped data set belong to the same connected component, their corresponding data points in the original data set are in the same group. Through such a mapping, we obtain the initial groups of the original data set. The number of the initial groups before merging is equal to the number of the connected components of the *1*-mutual graph, which is less than $O(\sqrt{n})$, i.e., the size of the zipped data set.

The chief requirement on preparing initial groups is that the points of different natural clusters cannot be in the same initial group, i.e., there must be no bridge between different clusters. This is exactly the aforementioned third characteristic of GraphZip, which is guaranteed by constructing a *1*-mutual graph on the zipped data set. The following theorem explains the reason.

Theorem 3.2 Neither one-segment bridge nor multi-segment bridge exists in the constructed *1*-mutual graph.

Proof. Assume a bridge $(V_1, V_2, ..., V_x)$, where V_1 belongs to cluster A and V_x belongs to cluster B. For multi-segment bridge, illustrated in Fig. 5(a), we have $x>2$. Let $d(V_i, V_{i+1})$ denote the distance between two consecutive vertices along the bridge, then we have a set of distances T: $d(V_1, V_2)$, $d(V_2, V_3)$,..., $d(V_{x-1}, V_x)$. Without losing generality, suppose the longest distance in T is the one between V_t and V_{t+1}, $1 \le t < x$. There are three cases for different t: 1) if $t=1$, then $d(V_{t+1}, V_{t+2}) < d(V_t, V_{t+1})$; 2) if $t=x-1$, then $d(V_{t-1}, V_t) < d(V_t, V_{t+1})$; 3) if $1<t<x-1$, both $d(V_{t+1}, V_{t+2})$ and $d(V_{t-1}, V_t)$ are smaller than $d(V_t, V_{t+1})$. According to the definition of k-mutual graph, in any of the 3 cases edge (V_t, V_{t+1}) does not exist in the *1*-mutual graph, so the bridge $(V_1, V_2, ..., V_x)$ is broken.

Fig. 5(b) shows an example of one-segment bridge, i.e., $x=2$. According to the definition of k-mutual graph, there are at most k edges connected to each vertex. So if the edge (V_1, V_2) exists in the sub-graphs created by the *1*-mutual graph, both V_1 and V_2 are isolated in clusters A and B, i.e., A and B are separated; if (V_1, V_2) does not exist, the bridge does not exist. □

(a) **(b)**

Figure 5. (a) The multi-segment bridge and (b) one-segment bridge

Bridge breaking issue has been well known since the proposal of single-linkage clustering algorithm such as MST [21, 5]. Many clustering algorithms suffer from the two types of bridges while our approach does not.

4. A HIERARCHICAL CLUSTERING METHOD

This section will present the hierarchical combination criterion. The basic idea of a hierarchical merging is as follows: we continuously choose the most appropriate pair of points from the initial groups and merge the chosen pair until reaching one cluster. At each hierarchical level a value M is computed for each pair of groups, denoted by $M(i,j)$ for groups i and j. The hierarchical process merges the pair of groups that maximizes M at each hierarchical level until all the groups have been merged, and there is a combination criterion to specify how to compute M. As the hierarchical process continues, the number of groups decreases by *1* at each hierarchical level. In our approach, the merging decision is made based on a k-mutual graph constructed on the original graph. Each initial group of the original data set corresponds to a set of vertices in the k-mutual graph. Suppose the k–mutual graph is $G=(V, E)$ and S_1, S_2, ..., S_t are sets of vertices corresponding to the t initial groups. We denote the number of edges between S_i and S_j as $E(i, j)$, and the size of S_i is defined as the number of vertices in S_i, denoted by $|S_i|$. The proposed combination criterion consists of two formulas:

$$M(i,j)=E(i,j)^2/MAX(|S_i|,|S_j|) \quad (1)$$

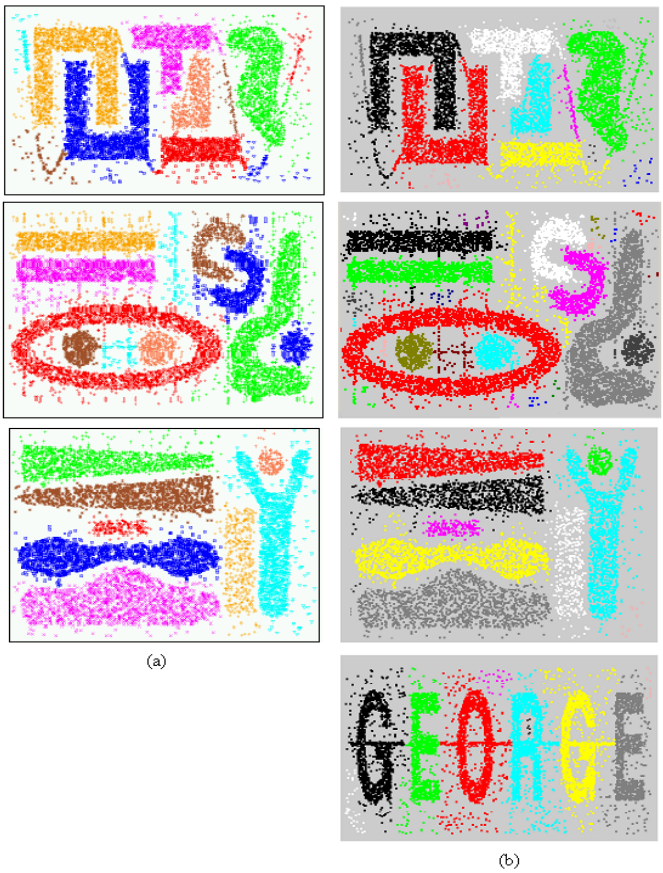
$$M(i,j)=E(i,j)^2/MIN(|S_i|,|S_j|) \quad (2)$$

Both formulas favor the number of connections between two groups over their sizes. The more connections, the more likely the two groups will be merged. On the other hand, if the connection is the same for two pairs of groups, using formula (1) will merge the pair containing the biggest group later than the other, while using formula (2) will merge the pair containing the smallest group first. Formula (1) does not favor adding points to a big group while formula (2) favors it as long as the number of the points being added is small enough. Thus Formula (1) can be used to produce big groups by merging small ones, once a group has been big enough, it cannot absorb any more points with formula (1), while formula (2) can add small groups of points to it continuously.

The hierarchical process using the proposed combination criterion is as follows: in the lower half of the hierarchy, i.e., when the number of groups is greater than $t/2$ for t initial groups, formula (1) is used; at higher hierarchical levels, i.e., when the number of groups is smaller than or equal to $t/2$, we use formula (2).

The reason of using such a hybrid criterion is due to noise. As a rule of thumb, noise may hinder the merging process. At the beginning of the hierarchical merging process many of the small groups are noise and they have similar connections with the groups of true data, using formula (2) may wrongly merge the noise and the true data. In such a case, we propose formula (1) to produce big groups first. Formula (1) takes shapes of the clusters by merging closely related groups first, and then Formula (2) retouches the formed big groups by adding smaller groups into them. According to the definition of Formula (2), the small groups will be merged into the big groups in an order according to the number of connections. A group of true data will be merged into the big groups earlier than a group of noise since noise is not closely related to the clusters as true data. Using Formula (2) can delay the merging of two natural clusters until all parts of each cluster have been merged. The experimental studies have supported this design strategy.

5. EXPERIMENTAL RESULTS



(a) (b)

Figure 6. The clustering results of (a) CHAMELEON; (b) GraphZip

Our experimental study is conducted on the four benchmark data sets used in CHAMELEON. In the proposed approach, the first step is GraphZip, which is parameter-free and independent of the dimensionality of the data. This step compresses the size of the original data to $O(\sqrt{n})$. After GraphZip, the following two steps are: constructing a *1*-mutual graph to obtain groups in the zipped data and mapping the grouping information back to the original data. Both steps are also parameter-free and independent of the dimensionality of the data. The last step constructs a *k*-mutual graph for the original data set for computing the merging criterion. We let k={20,30,40,50,60,70,80,90,100} and find that the proposed hierarchical merging criterion produces the same merging result for different *k* with the four data sets. Thus we simply fix the value of *k* in the program. In summary, the whole process can be completed automatically, and the preprocessing steps, GraphZip and the *1*-mutual graph construction, are general and efficient enough to be applicable to other graph-theoretical clustering algorithms.

Since CHAMELEON has experimentally outperformed ROCK, CURE, and DBSCAN on the cluster quality, this section only compares our results with CHAMELEON. The final clustering results of the four data sets are illustrated in Fig. 6, where each cluster has its own gray level/color. The results of CHAMELEON are on the left ones, i.e., Fig. 6 (a), and the results for the same data sets produced by our approach are on the right, i.e., Fig. 6 (b). Since CHAMELEON has not provided the result of the fourth data set in their paper, we list our result for it separately. For the first three data sets, our approach discovers all the natural clusters correctly. The results are very similar to the ones of CHAMELEON while our approach is faster and automatic.

6. CONCLUSION

This paper has presented a novel spatial data compression method, called GraphZip, which can reduce the size of the original data set significantly. The most important characteristic of GraphZip is that it can preserve the spatial patterns of the original data set in the compressed data, which makes GraphZip a very promising preprocessing method for existing clustering approaches. Besides, GraphZip is efficient and parameter-free.

Since data compression usually causes information loss, the quality of the resulting clusters are often compromised. However, the steps following GraphZip can minimum such a loss by mapping back the preliminary grouping information to the original data set before the real clustering process begins. In other words, clustering is still performed on the original data set while the closely related data has been grouped together so that the clustering process is dramatically simplified and thus much faster. The experimental studies on four benchmark spatial data sets produce comparable results with those of CHAMELEON, a state-of-the-art spatial clustering method, while our approach requires only $O(n\log n)$ time.

Future research will concentrate on studying (1) the compression levels of GraphZip for different data sets, (2) the effectiveness of combining GraphZip with existing clustering methods, and (3) corresponding noise and outlier handling mechanisms.

7. REFERENCES:

[1] Bentley, J. L. (1975). Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM* 18(9), pp. 509-517.

[2] Clarkson, K. L. (1983). Fast algorithms for the all nearest neighbors problem. *Proc. of 24th Annual Symposium on Foundations of Computer Science*, pp. 226-232.

[3] Ertoz, L., Steinbach, M., and Kumar, V. (2003), Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data, *Proc. of SIAM DM03*.

[4] Ester, M., Kriegel, H. P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise, *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD-96)*, AAAI Press, pp. 226-231.

[5] Estivill-Castro, V. and Lee, I. (2000). AUTOCLUST: Automatic clustering via boundary extraction for mining massive point-data sets, *5th Int'l Conf. on Geocomputation*, Geo Computation CD-ROM: GC049, ISBN 0-9533477-2-9.

[6] Guha, S., Rastogi, R., and Shim, K. (1998). CURE: An efficient clustering algorithm for large databases, *Proc. 1998 ACM SIGMOD Int. Conf. Management of Data*, pp. 73-84.

[7] Guha, S., Rastogi, R., and Shim, K. (1999). ROCK: a robust clustering algorithm for categorical attributes. *Proc. 1999 Int. Conf. on Data Eng.*, pp. 512-521.

[8] Han, J. and Kamber, M. (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers.

[9] Han, J., Kamber, M., and Tung, A. K. H. (2001). Spatial clustering methods in data mining: A survey, H. Miller and J. Han (eds.), *Geographic Data Mining and Knowledge Discovery,* Taylor and Francis.

[10] Harel, D. and Koren, Y. (2001). Clustering spatial data using random walks, *Proc. 7th Int'l Conf. Knowledge Discovery and Data Mining (KDD-2001)*, ACM Press, New York, pp. 281-286.

[11] Karypis, G., Han, E., and Kumar, V. (1999). CHAMELEON, A hierarchical clustering algorithm using dynamic modeling, *IEEE Computer*, Vol.32, pp. 68-75.

[12] Koperski, K., Han, J., and Adhikari, J. (1998). Mining Knowledge in Geographical Data. *Communications of the ACM*, 26(1), pp. 65-74.

[13] McCallum, A. Nigam, K., and Ungar, L. (2000). Efficient clustering of high-dimensional data sets with application to reference matching. *Proc. of KDD2000*, ACM Press, pp. 169-178.

[14] McQueen, J. (1967). Some methods for classification and analysis of multivariate observations, In *Proc. of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp.281-297.

[15] Moore, A. (1999). Very fast EM-based mixture model clustering using multiresolution *kd*-trees. *Advances in Neural Information Processing Systems*, Vol. 11 (1999), Morgan Kaufman, pp. 543--549.

[16] Ng, R., and Han, J. (1994). Efficient and effective clustering method for spatial data mining. *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB94)*, pp. 281-297.

[17] Smid, M. (1997). Closest-Point Problems in Computational Geometry. *Handbook on Computational Geometry*. J. R. Sack and J. Urrutia, editors, Elsevier Science Publishing, pp. 877-935.

[18] Vaidya, P. M., (1989). An O(nlogn) algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry* 4 (1989), pp. 101-115.

[19] Wang, W., Yang, J., and Muntz, R. (1997). STING: A Statistical Information Grid Approach to Spatial Data Mining, *Proc. of the 23rd Int. Conf. on Very Large Data Bases,* Morgan Kaufmann, pp. 186-195.

[20] Xu, X., Ester, M., Kriegel, H. P., and Sander, J. (1997). Clustering and knowledge discovery in spatial databases. Vistas in Astronomy, 41(1997), pp. 397-403.

[21] Zahn, C. T. (1971) Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters, *IEEE Trans. Comput.* C-20 (Apr. 1971), pp. 68-86.

[22] Zhang, T., Ramakrishnan, R., and Linvy, M. (1996). BIRCH: an efficient data clustering method for very large databases, *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, ACM Press, pp.103-114.