Clustering Large Dynamic Datasets Using Exemplar Points

William Sia, Mihai M. Lazarescu

Department of Computer Science, Curtin University,
GPO Box U1987, Perth 6001, W.A.
Email: {siaw, lazaresc}@cs.curtin.edu.au

Abstract. In this paper we present a method to cluster large datasets that change over time using incremental learning techniques. The approach is based on the dynamic representation of clusters that involves the use of two sets of *representative* points which are used to capture both the current shape of the cluster as well as the trend and type of change occuring in the data. The processing is done in an incremental point by point fashion and combines both data prediction and past history analysis to classify the unlabeled data. We present the results obtained using several datasets and compare the performance with the well known clustering algorithm CURE.

1 Introduction

Clustering is the process of discovering sets of similar objects and grouping them into classes based on similar patterns. Existing clustering algorithms can be grouped into several categories with each category attempting to address a different set of problems. Most current algorithms use ***static models***. Hence, they generally have a poor performance when confronted with sets of data that do not fit with their choice of static models. This happens in particular when dealing with data that contains clusters of varying sizes, shapes and attributes.

Another significant problem is that many of the current algorithms ***do not scale well*** when handling large data sets. Furthermore, the majority of the existing algorithm are designed to work with static datasets. The work we present in this paper attempts to address the problems of *scalability* and the *static models*. We also introduce a further requirement: that of processing dynamic datasets.

The term ***dynamic*** means that the data representation changes from time to time. There are two reasons for focusing on dynamic datasets. The first is that many of the existing data mining applications involve datasets which change over time while the second reason is the fact that in many domains extracting information about the trend and the changes that occur in the clusters/classes provides crucial clues on how the information should be interpreted (e.g. medical studies that are used to identify the changes conditions that can trigger the onset of a particular mental/behavioural problem). Most current algorithms do not reflect or interpret these changes in their computation as they merely produce a snapshot of the data. In this paper, the proposed approach detects the changes that occur and tries to incorporate this information to predict future data. As a result, it is able to represent the data dynamically.

We use a hierarchical approach to group the data that uses a subset of representative points to define the clusters discovered. The processing is done incrementally. However, unlike previous clustering methods, we use a number of machine learning techniques derived from drift tracking techniques which enable our algorithm to handle more effectively the dynamic aspect of the data.

The paper is organised as follows: Section 2 presents an overview of the previous research that is related to our work. Section 3 describes the algorithm while in Section 4 we present the two sets of results as well as a comparison with the CURE algorithm. The conclusions are covered in Section 5.

2 Related Work

Clustering is the process of segmenting data sets into groups of similar objects. It usually contains of two basic steps. The first step is to determine the number of possible clusters. Based on that number of clusters, the program tries to fit the best possible clustering[1]. Fields of study such as statistics, pattern recognition, and machine learning utilize clustering in their applications[2]. Clustering algorithms can be categorized into classes such as hierarchical , partitioning , grid-based and density based methods. In hierarchical clustering, a series of partitions take place before the optimal number of clusters is produced. Hierarchical clustering has the advantages of flexibility regarding the level of granularity. However the termination criteria is quite subjective in a sense that it differs from case to case. Moreover, the general approach does not perform any general refinement and improvement to clusters that have been constructed.

$BIRCH$ [3] (Balanced Iterative Reducing and Clustering using Hierarchies) is an example of an agglomerative hierarchical clustering algorithm that allows additional passes to improve the cluster quality. $BIRCH$ incrementally and dynamically clusters incoming multi-dimensional metric data points to try to produce the best quality clustering with the available resources. $CURE$ [4] is a hierarchical clustering algorithm that integrates the centroid-based and all-point extremes approach when representing a cluster. The number of points to represent a cluster is determined by a constant number *c* previously selected by the user. By using well scattered points, $CURE$ can handle clusters with varying sizes. Moreover, this approach results in a better performance compared to $BIRCH$ in discovering non-spherical clusters. $CURE$ allows the user to select the parameter α, that is used as a fraction when it tries to shrink the chosen scattered-points toward the centroid. These shrunken new points will reside between the centroid and the outline of the clusters. However, using representative points may not achieve an accurate result when selected points are insufficient to define the cluster representation. The problem is compounded when the shrinking phase eliminates some points that are ideal to represent the cluster. $CHAMELEON$ [5] solves this shortcoming by determining the pair of most similar sub-clusters by considering both the inter-connectivity (*RI*) as well as the closeness (*RC*) of the clusters. $CHAMELEON$ utilizes dynamic modeling in cluster aggregation. The algorithm works by finding the *K*-nearest neighbors for each point in the starting phase. Combined with the graph partitioning technique [6], this first stage produces small tight clusters. In the next stage, the algorithm performs the agglomerative process. It tries to merge the small tight clusters based on the relative inter-connectivity and relative closeness; both are locally normalized by quantifiers. The approach used by $CHAMELEON$ provides the solution for finding clusters with irregular shape, sizes and density. It achieves this by using dynamic modeling that requires the sets of data to be constructed as a sparse graph in the early stage. Constructing the sparse graph from the data has proven to be time consuming, and therefore inefficient to be used when processing large sets of data. However, this algorithm can be used as an initial stage in processing data incrementally. In our work, $CHAMELEON$ will be used to build the initial clusters. The best representation of the initial clusters is crucial because when processing a new data point, the clustering decision will be based on these clusters. Medoid-based approaches, such as $CLARANS$ [7] and PAM [8] try to find representative points (medoids) to minimize the sum of the distances of points from the closest medoid. This offers two advantages: (1) it solves the metric spaces-specific data problem faced by the centroid approaches and (2) it eliminates outliers by selecting the medoids based on the location of the predominant fraction of points inside a cluster. However, this method fails when dealing with natural data where points in a given cluster are closer to the center of another cluster than to the center of their own cluster. A clustering approach similar in some aspects to our work is used by $DEMON$ [9]. $DEMON$ is an algorithm that takes the temporal aspect of data evolution into account and allows an analyst to ”mine” relevant subsets of the data. This algorithm has the limitation that it uses the *windowing approach* when processing the new data. Furthermore, $DEMON$ assumes a supervised scenario (all data has been previously labeled) and it deals with only a restricted number of points.

3 Algorithm

The algorithm we have developed was initially based on CHAMELEON. The reason for selecting CHAMELEON was its ability to handle clusters of different shapes and sizes with high accuracy. Unfortunately, the processing done by CHAMELEON utilizes a *k-nearest neighbor approach* [10], which has two major drawbacks: it requires a significant amount of processing time and it does not scale well to large datasets. Furthermore, CHAMELEON does not incorporate incremental processing, hence the algorithm needs to reprocess old data points when the new data arrives. This makes it unsuitable for dynamic sets where the data is observed as a sequential stream of information. However, for our clustering algorithm to be effective, it is desirable to generate an initial set of clusters which closely mirror the groupings found in the data. Hence, our algorithm generates the initial set of clusters using the CHAMELEON approach. Once the initial set of clusters is developed, the processing involved in our approach is completely different from that of CHAMELEON.

3.1 Cluster Representation

We developed a data structure specifically designed to allow fast, incremental updates. The data structure has two levels. The first level stores all the data points observed while the second level stores only the summary definitions of the clusters. To reduce the computational complexity of the method most of the processing is done at the second level of the data structure which is stored in the memory. Fundamental to our work is the cluster representation. Each cluster representation contains three dynamic sets of points: an exemplar set, a predictor set and knowledge repository set. The exemplar points are used to define the basic summary representation of the cluster and each exemplar point has three attributes: reinforcement, age and coverage. All three attributes are dynamic and indicate the “contribution” of the exemplar to the cluster definition. The reinforcement is computed based on the number of new data points observed which have been similar to the exemplar. The age of the exemplar is defined by the time interval over which the exemplar has been consistently reinforced. The coverage is determined by an analysis of the distance between the exemplar and all other exemplars used to define the cluster. Apart from the three attributes, each exemplar has a link to a list of points which are associated with the exemplar at the first level of the data structure. Thus, if a very detailed analysis of the data previously observed is required, the algorithm only needs to retrieve the list from the disk. The exemplar points are extracted from the initial set of clusters generated by the algorithm. From each cluster discovered in the data, the algorithm extracts *k* points which best capture the shape and size of the cluster. The points are extracted based on reinforcement and spread (average distance to the other k points). The *k* points are sorted and the best 10 *k* points (that maximize both reinforcement and average distance) become the exemplar points representing the cluster. As more data becomes available the exemplar points representing the cluster may change to reflect the new trends in the data. In our work we have investigated a range of values for *k* before choosing *k=10*. The results from the experiments indicate that while larger values such *k=15* do not impact significantly on the algorithm processing, the accuracy gained from the extra points was not statistically significant. On the other hand, a smaller *k* value such as *k=5* was insufficient to represent some of the more unusual types of clusters that were used in the experiments. The novel aspect of the cluster representation is the use of *predictor* points and a knowledge repository. These two sets of points are used for three reasons: (1) to augment the cluster representation, (2) to adapt the cluster representation to any changes detected in the data and (3) to maintain a history of the information observed. The *predictor* points are used to augment the exemplar points that are used in the summary definition of the cluster. Their purpose is to allow the tracking of changes and prediction of future data. Initially, this set of points are the next best *k*-points aside from the *exemplar* points to represent the cluster. The selection of the *predictor* points is different from the *exemplar* selection. The requirements for the *predictor* points are that (1) the points need to capture the temporal aspect of the cluster representation and (2) the points should be located some distance between the existing exemplar and predictor points to ensure a good spread of the data representation. Each new data point observed is a potential predictor point candidate and the algorithm has an in-built preference for selecting points which are positioned near the border of the cluster. This was done specifically to allow the close tracking of any changes found in the data. The drawback of this approach is that in cases where the data is noisy the predictors are frequently changed before a reliable subset is found. In general, however, the approach is effective as the clusters were accurately updated to mirror the changes in the data. The predictor selection is done as follows. Each time the algorithm reads a new data point, it calculates the distance between that point and the exemplar points and stores each of the calculation into the EX-DISTANCE-LIST. It also performs a similar calculation with the predictor points and stores the calculation into the PRED-DISTANCE-LIST. From both the EX-DISTANCE-LIST and PRED-DISTANCE-LIST, the program retrieves the minimum value, which indicates the minimum distance between the new point to the exemplar points and the predictor points. If the minimum distance is less than the average distance of the cluster, the new data point should be considered as a predictor point. Using this approach conforms to the two requirements above. First, because the candidate point is chosen from the new data point, it is able to capture the temporal information of the cluster. Second, performing the distance checking between the predictor and the exemplar points guarantees the widespread of the new point. The *knowledge repository* has the purpose of storing old data knowledge in the form of "old predictor and old exemplar points". These points can be used to effectively deal with a recurrent trend in the data. It also reduces the complexity of the processing because we do not need to rediscover clusters and cluster information which existed in the past. Lastly, by keeping the old cluster representation, we are able to produce a continuous interpretation of the cluster definition from time to time.

3.2 Incremental Processing

Most of the processing takes place at the second level of the structure and the processing is of an incremental nature (point by point). The assumptions made were as follows: (1) an initial training set is available at the start of processing and (2) the data observed after the training stage takes the form of a sequential stream of points. The approach we use is unsupervised and all the data processed is assumed to be unlabeled. In the training stage the system generates the set of initial clusters using the CHAMELEON approach. The clusters are processed and the exemplar and predictor points are extracted for each cluster. After the training stage is completed, the algorithm uses incremental learning to handle the rest of the data observed. The program reads the new data points and tries to assign the points to existing clusters if they satisfy all the necessary requirements. The requirements are based on a comparison of the distance between the new data point to the cluster's representation points and the average distance between the representation points. When locating the closest cluster, the general approach is to assign new data points to a cluster according to the closest *k* nearest exemplar. An average distance between the *k* nearest exemplar and the new point is derived for each set of clusters and, the cluster with the smallest average distance between the exemplar and the point is the cluster the new point is most likely to be assigned to. If the distance between the new point and the exemplar or the centroid is greater than the average distance between points in a cluster, the algorithm assigns the new point as the new cluster. Next, it will update the reinforcement of the exemplar point and the predictor point lists. The algorithm attempts to match the new point to a predictor or an exemplar of an existing cluster. The point can either reinforce an existing exemplar or predictor point or form the basis of a new cluster. As the data is observed, we use the set of predictor and exemplar points to update the cluster definition. Over time, the points in the predictor list may become more significant than the exemplar points. The algorithm updates the representation points by considering the predictor points as candidate points. First, it inserts the points from the predictor list that meet *reinforcement threshold* into the exemplar point list. Next, it retires the exemplar points that are no longer important into the repository list. The retirement of an exemplar point is based on both the *reinforcement* and the *age* attributes of the exemplar.

The predictor point update is similar to the exemplar point processing. A predictor point's reinforcement value is updated when that point is located closer to the new point when compared with the closest exemplar. The algorithm searches for a point which has the closest distance to the new point. If such a point exists, then the algorithm updates its reinforcement value. Updating the reinforcement information of the predictor point is essential because it indicates the importance of that point in a particular time frame. If the level of reinforcement of a predictor point is low then the point is "retired" in the knowledge repository. On the other hand if the level of reinforcement is high, a predictor can be promoted to an exemplar point. The reinforcement of points is driven by the underlying trend in the data. In some cases, if the trend in the data is recurrent, old points from the knowledge repository may be reused in cluster definitions. The algorithm always checks the repository for whenever a new predictor point is added to the predictor list. If the new candidate is similar to a past predictor/exemplar, then the old point is reused instead of the new candidate. After the reinforcement update, the program checks for special cases on the current cluster representation. Hence the algorithm checks whether it should merge two clusters or split the current cluster. If the new data point is not assigned to any clusters, then a new cluster is created.

4 Results

The algorithm was tested using several generated datasets. The datasets were created using VidGen, a tool that allows the user to specify the shape, size and density of a cluster as well as the number of clusters to be generated for the dataset. Furthermore, the data had a time index which allowed the simulation of (1) different trends over time and (2) different reinforcement patterns. In this section we present detailed results from 2 scenarios. Each scenario consisted of a training set (from which the initial set of clusters is generated) and test set (where the data arrives in the form of a sequential stream) ranging from 100,000 to 2,000,000 points. The test set was further divided into several consecutive phases to allow for a more detailed analysis of the algorithm performance (hence we present the results for each individual phase in the test set). The aim of each scenario was to test the different requirements of the algorithm: accuracy, data prediction, and computation efficiency. All the results presented have been obtained after 10 runs and an average over the 10 runs is given as the final result. The PC used for the experiments was a P4-1.8Mhz with 256Mb of RAM running Windows2000.

4.1 Scenario 1

The scenario was designed to test the speed and clustering accuracy of the program when dealing with large datasets, specifically 2,000,000 points. The test was carried out on 3,000 training data points and 2,000,000 incoming data points depicted in Figure 1. The incoming data arrived in three phases. The accuracy of the clustering algorithm and the time of processing will be shown in table for each phases. The table explanation is as

	Training Set		Combined Sets Phase One		
	Original	New Approach	Original	Added Point	New Point
Cluster 1	1135	1135	1135	-	1135
Cluster 2	1095	1095	1095	-	1095
Cluster 3	822	822	822	-	822
Cluster 4	-	-	-	691197	689062
Dropped	-	-	-	-	-
Time	-	-	-	-	5 mins 13 secs

Table 1. Test With Large Data Set - First Phase

	Combined Sets Phase Two		
	Original	Added Point	New Point
Cluster 1	1135	-	1135
Cluster 2	1095	681087	682182
Cluster 3	882	-	882
Cluster 4	691197	-	689062
Dropped	-	-	-
Time	-	-	11 mins 28 secs

Table 2. Test With Large Data Set - Second Phase

	Combined Sets Phase Three		
	Original	Added Point	New Point
Cluster 1	1135	-	1135
Cluster 2	682182	-	682182
Cluster 3	882	667647	668529
Cluster 4	691197	-	689062
Dropped	-	-	-
Time	-	-	16 mins 26 secs

Table 3. Test With Large Data Set - Third Phase

follows. The *training set* column contains the information about the training stage and is divided into two sub-columns. The *original* sub-column indicates the actual number of data points of each cluster while the *new approach* sub-column indicates the number of data points that is clustered correctly to the cluster. The *combined sets phase one* column contains the result produced in phase one and is itself divided into three sub-columns. The *original* sub-column indicates the number of data points of each cluster before the new data arrives. The *added point* sub-column indicates the number of data points added to a particular cluster. Lastly, the *new approach* sub-column contains the number of points of each cluster after the new data points are added. The tables in the other results sub-sections of the paper follow the same format described above.

Phase One: In the first phase, the incoming data arrives on the area that contains no existing clusters (see Figure 2). The algorithm creates a new cluster and classifies all the incoming data into single cluster. The processing time to classify the first phase is five minutes and thirteen seconds.

Phase Two: In the second phase, the data arrives on an area near cluster two. Figure 3 illustrates the current exemplar points when all the new data points have been classified. The total time required to process the first and second phase is eleven minutes and twenty eight seconds.

Phase Three: In the final phase, the data arrives on area near cluster three. Figure 4 illustrates the current exemplars of cluster three after the cluster representation is updated. The overall time to classify the three phases is sixteen minutes and twenty six seconds.

5 Scenario 2 - Comparison with CURE

Scenario two was designed to compare our approach with an existing algorithm. We selected CURE for two reasons. It is an algorithm that generally has good performance in

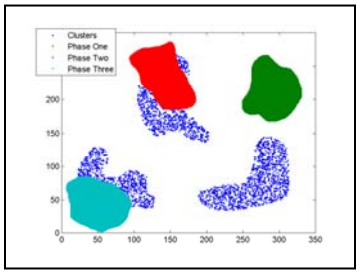

Fig. 1. Combined Phases - Training & Test Data

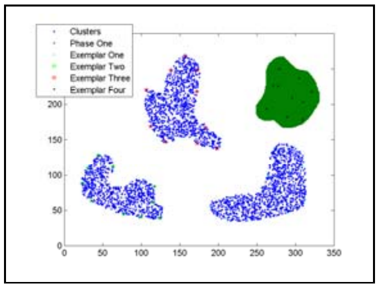

Fig. 2. Phase One - Training & Test Data.

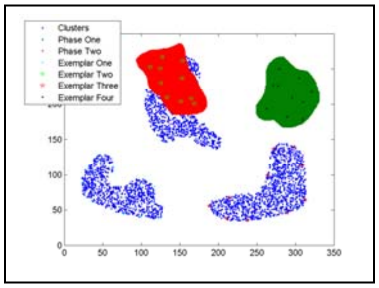

Fig. 3. Phase Two - Training & Test Data.

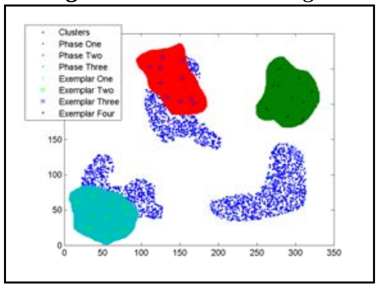

Fig. 4. Phase Three - Training & Test Data.

terms of handling data with clusters of different shapes and sizes, as well as in terms of speed. We also intended to compare our approach with CHAMELEON but the memory and computational requirements of the CHAMELEON made testing with large datasets unfeasible. CURE does the clustering process in one-phase while our approach processes data incrementally. As a result, the test cases are designed to fulfill the requirement of the two algorithms. We did the testing using two sets of data, starting from 20,000 points to 100,000 points. Figure 5 illustrates the training data and the incremental data used in testing. The shape of the clusters throughout the testing is the same. The input data for CURE is the combination of the training data and the incremental data. For our approach, we start by processing the training data and then incrementally process the rest of the test data. Figure 6 illustrates the exemplar points extracted from the clusters.

Testing Using 20,000 Points Figure 7 illustrates the result produced by CURE. When compared with our approach, CURE fails to adjust the cluster definition to accurately reflect trend shown by the new data points and this reduces the CURE overall accuracy (most of the new points are missclassified). This result is not surprising because the size of the incremental data is larger than the training, and CURE prefers to select clusters that have high density. On other hand, our algorithm classifies the data points with high accuracy. Our algorithm does not rely heavily on the data density. It considers density as one of the trends of the data, so data can have many levels of density in different time frames. Figure 8 illustrates the result produced using our approach. Table 5 summarizes the overall result. Finally, our method is faster when compared with CURE.

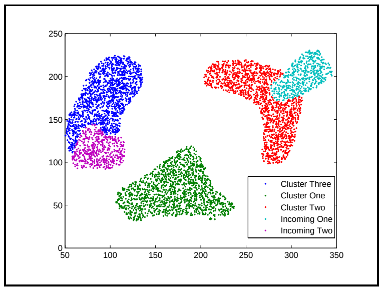

Fig. 5. Scenario 2 - Test & Training Set Combined.

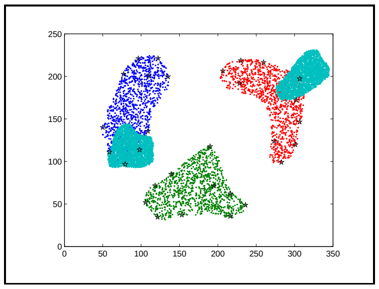

Fig. 6. Scenario 2 - Exemplar Points Extracted by Our Approach.

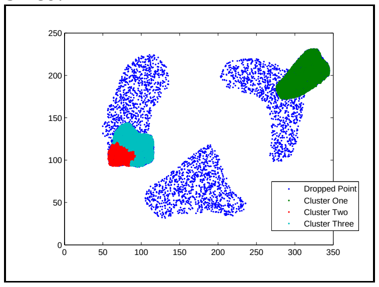

Fig. 7. CURE - 20,000 Points Test.

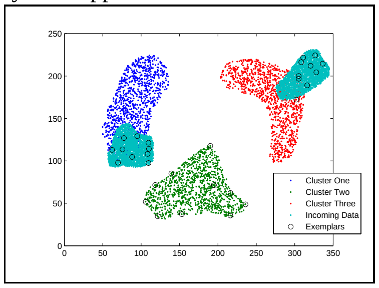

Fig. 8. Our Approach - 20,000 Points Test.

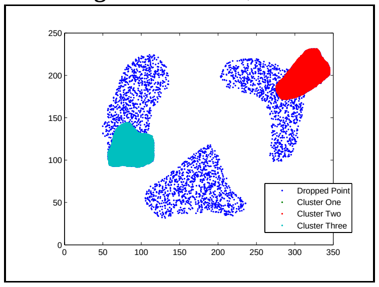

Fig. 9. CURE - 100,000 Points Test.

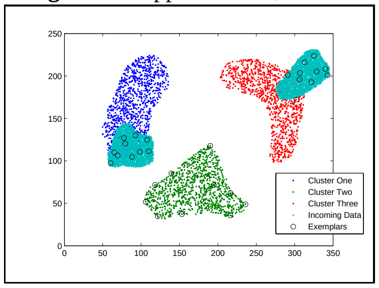

Fig. 10. Our Approach - 100,000 Points Test.

	Combined Sets 20000 Test		
	Original	Cure	Our Approach
Cluster 1	11019	2728	10895
Cluster 2	11026	5765	10934
Cluster 3	1071	0	1071
Time	-	7 mins 23 secs	47 secs
Dropped	-	14623	-

Table 4. Scenario 2 Result - 20000 Test.

	Combined Sets 100000 Test		
	Original	Cure	Our Approach
Cluster 1	50784	37913	49784
Cluster 2	51774	38366	50374
Cluster 3	1071	0	1071
Time	-	2 hrs 10 mis	1 min 3 secs
Dropped	-	25208	-

Table 5. Scenario 2 Result - 100000 Test

Testing Using 100000 Points The second test was carried out using 100,000 points. The same approach was applied where the size of the testing data remains the same while we increase the size of the incremental data. The result is similar to the previous test (see Figures 9 and 10) . However, there is one more major limitation that is not addressed by CURE. In Table 5, we can see that in it takes two hours and thirteen minutes to process 100,000 data points, which makes CURE unsuitable to handle large data sets. Our approach deals only with the exemplar and predictor points. As a result it is computationally less intensive, because the processing is done incrementally rather than simultaneously.

6 Conclusions

In this paper we have presented an algorithm to cluster large dynamic datasets. The research has two novel aspects: the use of new cluster representation which combines both exemplar and predictor points and the integration of drift tracking techniques in the incremental processing of the data. The algorithm has been tested with several synthetic datasets and its performance was compared with the CURE algorithm. The results show the algorithm has good accuracy and it is both faster and more acurate when compared with CURE. Future work will be done on improving the overall speed of the algorithm and on using more advanced tracking techniques to allow for a better adaptation of the clusters to changes in the data.

References

1. Bradley, P.S., Fayyad, U.M., Mangasarian, O.L.: Data mining: Overview and optimization opportunities. Technical report, Microsoft Research Lab (1998)
2. Berkhin, P.: Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA (2002)
3. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: A new data clustering algorithm and its applications. Data Min. Knowl. Discov. **1** (1997) 141–182
4. Guha, S., Rastogi, R., Shim, K.: CURE: an efficient clustering algorithm for large databases. In: Proceeding of ACM SIGMOD International Conference on Management of Data, Seattle, WA, USA (1998) 73–84
5. Karypis, G., Han, E.H.S., Kumar, V.: Chameleon: Hierarchical clustering using dynamic modeling. Computer **32** (1999) 68–75
6. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20** (1998) 359–392
7. Ng, R.T., Han, J.: Clarans: A method for clustering objects for spatial data mining. IEEE Transactions on Knowledge and Data Engineering **14** (2002) 1003–1016
8. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons, New York (1990)
9. Ganti, V., Gehrke, J., Ramakrishnan, R.: Demon: Mining and monitoring evolving data. IEEE Transactions on Knowledge and Data Engineering **13** (2001) 50–63
10. Therrien, C.W.: Decision estimation and classification: an introduction to pattern recognition and related topics. John Wiley & Sons, Inc. (1989)