Determining the Number of Clusters/Segments in Hierarchical Clustering/Segmentation Algorithms

Stan Salvador and Philip Chan
Dept. of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901
{ssalvado, pkc}@cs.fit.edu

ABSTRACT

Many clustering and segmentation algorithms both suffer from the limitation that the number of clusters/segments are specified by a human user. It is often impractical to expect a human with sufficient domain knowledge to be available to select the number of clusters/segments to return. In this paper, we investigate techniques to determine the number of clusters or segments to return from hierarchical clustering and segmentation algorithms. We propose an efficient algorithm, the L method, that finds the "knee" in a '# of clusters vs. clustering evaluation metric' graph. Using the knee is well-known, but is not a particularly well-understood method to determine the number of clusters. We explore the feasibility of this method, and attempt to determine in which situations it will and will not work. We also compare the L method to existing methods based on the accuracy of the number of clusters that are determined and efficiency. Our results show favorable performance for these criteria compared to the existing methods that were evaluated.

Keywords: clustering, cluster validation, segmentation, hierarchical clustering, time series.

1. INTRODUCTION

While clustering and segmentation algorithms are unsupervised learning processes, users are usually required to set some parameters for these algorithms. These parameters vary from one algorithm to another, but most clustering/segmentation algorithms require a parameter that either directly or indirectly specifies the number of clusters/segments. This parameter is typically either *k*, the number of clusters/segments to return, or some other parameter that indirectly controls the number of clusters to return, such as an error threshold. Setting these parameters requires either detailed pre-existing knowledge of the data, or time-consuming trial and error. The latter case still requires that the user has sufficient domain knowledge to know what a good clustering "looks" like. However, if the data set is very large or is multi-dimensional, human verification could become difficult. To find a reasonable number of clusters, many existing methods must be run repeatedly with different parameters, and are impractical for real-world data sets that are often quite large.

We desire an algorithm that can efficiently determine a reasonable number of clusters/segments to return from any hierarchical clustering/segmentation algorithm. In order to identify the correct number of clusters to return from a hierarchical clustering/segmentation algorithm, we introduce the L method. The definition of a "cluster" is not well-defined, and measuring cluster quality is subjective. Thus, there are many clustering algorithms with unique evaluation functions and correspondingly unique notions of what a good cluster "looks" like. The L method makes use of the same evaluation function that is used by a hierarchical algorithm during clustering or segmentation to construct an evaluation graph where the *x*-axis is

the number of clusters and the *y*-axis is the value of the evaluation function at *x* clusters. The knee, or the point of maximum curvature of this graph, is used as the number of clusters to return. The knee is determined by finding the area between the two lines that most closely fit the curve. The L method only requires the clustering/segmentation algorithm to be run once, and the overhead of determining the number of clusters is trivial compared to the runtime of the clustering/segmentation algorithm.

Our main contributions are: (1) we demonstrate a method to efficiently determine a reasonable number of clusters to return from any hierarchical clustering or segmentation algorithm; (2) we introduce a novel method to find the knee of a curve; (3) we explore the feasibility of using the "knee" of a curve to determine the number of clusters to return by evaluating its performance on several different algorithms and data sets; (4) we compare our method with two existing techniques: the Gap statistic and Permutation tests based on the accuracy of the number of clusters returned and efficiency.

The next section gives an overview of related work. Section 3 provides a detailed explanation of our L method, which is able to efficiently and accurately determine a good number of clusters to return from hierarchical clustering/segmentation algorithms. Section 4 discusses experimental evaluations of the L method and two existing methods using several clustering and segmentation algorithms, and Section 5 summarizes our study.

2. RELATED WORK

Clustering Algorithms. Clustering is an unsupervised machine learning process that creates clusters such that data points inside a cluster are close to each other, and also far apart from data points in other clusters. There are four main categories of clustering algorithms: partitioning, density-based, grid-based, and hierarchical. Partitioning algorithms, such as *K*-means and PAM [14], iteratively refine a set of *k* clusters and do not scale well for larger data sets. Density-based algorithms, e.g., DBSCAN [3] and DENCLUE [9], are able to efficiently produce clusters of arbitrary shape, and are also able to handle outliers. If the density of a region is above a specified threshold, those points are assigned to a cluster; otherwise they are considered to be noise. Grid-based algorithms, such as WaveCluster [17], reduce the clustering space into a grid of cells, enabling efficient clustering of very large datasets. Hierarchical algorithms can be either agglomerative or divisive. The agglomerative (bottom-up) approach repeatedly merges two clusters, while the divisive (top-down) approach repeatedly splits a cluster into two. CURE [6] and Chameleon [10] are examples of two hierarchical clustering algorithms. Hierarchical algorithms differ chiefly in the criteria that they use to determine similarity between clusters.

Segmentation Algorithms. Segmentation algorithms take time series data as input and produce a Piecewise Linear Approximation (PLA) as output. A PLA is a set of consecutive line segments that approximate the original time series. Segmentation algorithms are related to clustering algorithms in that each segment can be though of as a cluster. However, segmentation algorithms typically create a finer grain partitioning than clustering algorithms. There are three approaches to time series segmentation [11]. First, in the sliding window approach, a segment is grown until the error of the line is above a specified threshold, then a new segment is started. Second, in the top-down approach, a segment representing the entire time series is repeatedly split until the desired number of segments or an error threshold is reached. Third, the bottom-up approach typically starts off with $n/2$ segments, and the two most similar adjacent segments are merged until the desired number of segments or the error threshold is reached. Gecko [16] is a bottom-up segmentation algorithm that merges segments based on slope and creates an initial fine-grain approximation by first performing a top-down pass.

Determining the Number of Clusters/Segments. Five common approaches to estimating the dimension of a model (such as the number of clusters or segments) are: cross-validation, penalized likelihood estimation, permutation tests, resampling, and finding the knee of an error curve.

Cross-validation techniques create models that attempt to fit the data as accurately as possible. Monte Carlo cross-validation [18][17] has been successfully used to prevent over-fitting (too many clusters/segments). Penalized likelihood estimation also attempts to find a model that fits the data as accurately as possible, but also minimizes the complexity of the model. Specific methods to penalize models based on their complexity are: MML [1], MDL [8], BIC [5], AIC, and SIC [19]. Permutation tests [22] attempt to prevent the creation of a PLA that over-fits the data by comparing the relative change in approximation error to the relative change of a 'random' time series. If the errors are changing at a similar rate, then more segments would fit noise and not the underlying structure of the time series. Resampling [15] and Consensus Clustering [13] attempt to find the correct number of clusters by clustering many samples of the data set, and determining the number of clusters where clusterings of the various samples are the most "stable."

Locating the "knee" of an error curve, in order to determine an appropriate number of clusters or segments, is well known, but it is not a particularly well-studied method. There are methods that statistically evaluate each point in the error curve, and use the point that either minimizes or maximizes some function as the number of clusters/segments to return. Such methods include the Gap statistic [21] and prediction strength [20]. These methods generally (with the exception of hierarchical algorithms) require the entire clustering or segmentation algorithm to be run for each potential value of k.

The majority of these methods to determine the best number of clusters/segments may not work very well in practice. Model-based methods, such as cross-validation and penalized likelihood estimation, are computationally expensive and often require the clustering/segmentation algorithm to be run several times. Permutation tests and resampling are extremely inefficient, since they require the entire clustering algorithm to be re-run hundreds or even thousands of times. The majority of existing methods to find the knee of an error curve require the clustering algorithm to be re-run for every potential value of *k*. Even worse, many of the evaluation functions that are used to evaluate a set of clusters run in $O(N^2)$ time. This means that it may take longer just to evaluate a set of clusters than it does to generate them. Most methods that find the knee of a curve also only work well when the clusters are well separated.

A few existing clustering algorithms have built-in mechanisms for determining the number of clusters. The TURN* [4][2] algorithm locates the knee of a curve by location the point where the 2nd derivative increases above a user specified threshold. A variant [2] of the BIRCH [23] algorithm uses a mixture of the Bayesian Information Criterion (BIC) and the ratio-change between inter-cluster distance and the number of clusters.

Finding the Knee of a Curve. The knee of a curve is loosely defined as the point of maximum curvature. The knee in a "# of clusters vs. evaluation metric" graph can be used to determine the number of clusters to return. Various methods to find the knee of a curve are:

1. The largest magnitude difference between two points.
2. The largest ratio difference between two points [2].
3. The first data point with a second derivative above some threshold value [3][4].
4. The data point with the largest second derivative [7].
5. The point on the curve that is furthest from a line fitted to the entire curve.
6. Our L-method, which finds the boundary between the pair of straight lines that most closely fit the curve.

This list is ordered from the methods that make a decision about the knee locally, to the methods that locate the knee globally by considering more points of the curve. The first two methods use only single pairs of adjacent points to determine where the knee is located. The third and fourth methods use more than one pair of points, but still only consider local trends in the graph. The last two methods consider all data points at the same time. Local methods may work well for smooth, monotonically increasing/decreasing curves. However, they are very sensitive to outliers and local trends, which may not be globally significant. The fifth method takes every point into account, but only works well for continuous functions, and not

curves where the knee is a sharp jump. Our L method considers all points to keep local trends or outliers from preventing the true knee to be located, and is able to find knees that exist as sharp jumps in the curve.

3. APPROACH

Our L method attempts to determine an appropriate number of clusters/segments to return, by finding the knee in an evaluation graph. The next section describes evaluation graphs, followed by sections that describe the L method and refinements to it.

3.1 Evaluation Graphs

The information required to determine an appropriate number of clusters/segments to return is contained in an evaluation graph that is created by the clustering/segmentation algorithm. The evaluation graph is a two-dimensional plot where the *x*-axis is the number of clusters, and the *y*-axis is a measure of the quality or error of a clustering consisting of *x* clusters. Since hierarchical algorithms either split or merge a pair of clusters at each step, all clusterings containing '*1*' to '*the number of clusters in the fine-grain clustering*' clusters can be produced by running the clustering algorithm only once.

The *y*-axis values in the evaluation graph can be any evaluation metric, such as: distance, similarity, error, or quality. These metrics can be computed globally or greedily. Global measurements compute the evaluation metric based on the entire clustered data set, such the sum of all pairwise distances between points in each cluster. The greedy method computes the evaluation metric by evaluating only the two clusters that are involved in the current merge or split of the hierarchical algorithm, rather than the entire data set.

'External' evaluation methods attempt to determine the number of clusters by evaluating the output of an arbitrary clustering algorithm. Each evaluation method has its own notion of cluster quality. Most external methods use pairwise-distance functions that are heavily biased towards spherical clusters. Such methods are unsuitable for clustering algorithms that have a different notion of cluster distance/similarity. For example, Chameleon [10] uses a similarity function that can produce complex non-spherical clusters. Therefore, the metric used in the evaluation graph should be the same as the metric used in the clustering algorithm.

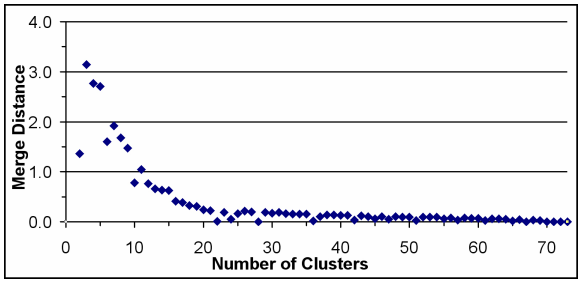

Figure 1. A sample evaluation graph.

An example of an evaluation graph produced by the Gecko segmentation algorithm [16] is shown in Figure 1. The *y*-axis values are the distances between the two clusters that are most similar at *x* clusters. This is a greedy approach, since only the two clusters being merged are used to generate the values on the *y*-axis. The curve in Figure 1 has three distinctive areas: a flat region to the right, a sharply-sloping region to the left, and a curved transition area in the middle.

In Figure 1, starting from the right, where the merging process begins at the initial fine grain clustering, there are many very similar clusters to be merged and the trend continues to the left in a rather straight line for some time. In this region, many clusters are similar to each other and should be merged. Another distinctive area of the graph is on the far left side where the merge distances grow very rapidly (moving right to left). This rapid increase in distance indicates that very dissimilar clusters are being merged together, and that the quality of the clusters is becoming poor because clusters are no longer internally homogeneous. A reasonable number of clusters is therefore in the curved area, or the "knee" of the graph. This knee region is between the low distance merges that form a nearly straight line on the right side of the graph, and the quickly increasing region on the left side. Clusterings in this knee region contain a balance of clusters that are both highly homogeneous, and also dissimilar to each other. Determining the number of clusters where this knee region exists will therefore give a reasonable number of clusters to return.

Locating the exact location of the knee, and along with it the number of clusters, would seem problematic if the knee is a smooth curve. In such an instance, the knee could be anywhere on this smooth curve, and thus the number of clusters to be returned seems imprecise. Such an evaluation graph would be produced by a data set with clusters that are overlapping or not very well separated. A time series usually represents continuous functions and are also not well separated. In such instances, there is no single 'correct' answer and all of the values along the knee region are likely to be reasonable estimates of the number of clusters. Thus, an ambiguous knee indicates that there probably is no single 'correct' answer, but rather a range of acceptable answers.

3.2 Finding the Knee via the L Method

In order to determine the location of the transition area or knee of the evaluation graph, we take advantage of a property that exists in these evaluation graphs. The regions to both the right and the left of the knee (see Figure 2) are often approximately linear. If a line is fitted to the right side and another line is fitted to the left side, then the area between the two lines will be in the same region as the knee. The value of the *x*-axis at the knee can then be used as the number of clusters to return. Figure 2 depicts an example.

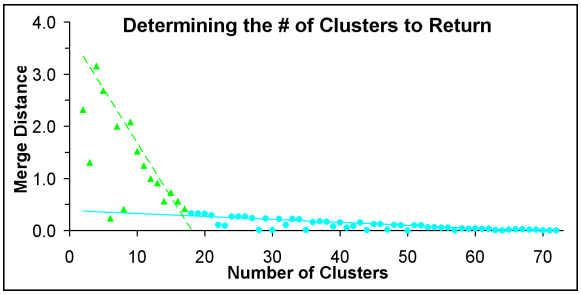

Figure 2. Finding the number of clusters using the L method.

To create these two lines that intersect at the knee, we will find the pair of lines that most closely fit the curve. Figure 3 shows all possible pairs of best-fit lines for a graph that contains seven data points (eight clusters were repeatedly merged into a single cluster). Each line must contain at least two points, and must start at either end of the data. Both lines together cover all of the data points, so if one line is small, the other is large to cover the rest of the remaining data points. The lines cover sequential sets of points, so the total number of line pairs is *numOfInitialClusters*-4. Of the four possible line pairs in Figure 3, the third pair fits their respective data points with the minimum amount of error.

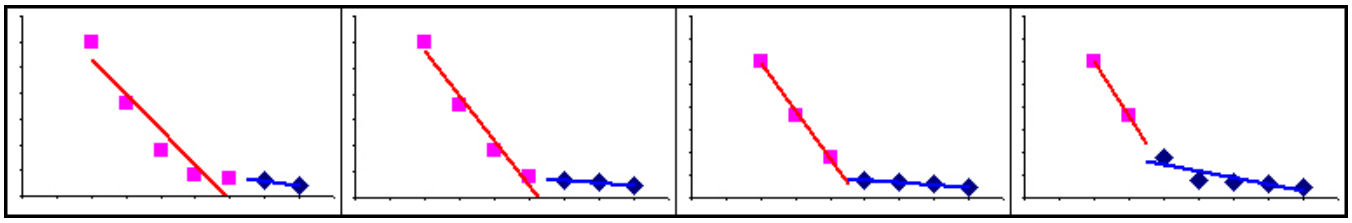

Figure 3. All four possible pairs of best-fit lines for a small evaluation graph.

Consider a '# of clusters vs. evaluation metric' graph with values on the *x* axis up to $x=b$. The *x*-axis varies from 2 to *b*, hence there are *b*-1 data points in the graph. Let L_c and R_c be the left and right sequences of data points partitioned at $x=c$; that is, L_c has points with $x=2...c$, and R_c has points with $x=c+1\ldots b$, where $c=3\ldots b-2$. Equation 1 defines the total root mean squared error $RMSE_c$, when the partition of L_c and R_c is at $x=c$:

$$RMSE_c = \frac{c-1}{b-1} \times RMSE(L_c) \; + \; \frac{b-c}{b-1} \times RMSE(R_c) \qquad [1]$$

where $RMSE(L_c)$ is the root mean squared error of the best-fit line for the sequence of points in L_c (and similarly for R_c). The weights are proportional to the lengths of L_c (c-1) and R_c (b-c). We seek the value of c, c^, such that $RMSE_c$ is minimized, that is:

$$c\hat{} = \arg\min_c \; RMSE_c \qquad [2]$$

The location of the knee at $x=c$^ is used as the number of clusters to return.

In our evaluation, the L method determined the number of clusters in only 0.00004% to 0.9% of the execution time required by the clustering algorithm. The time it takes for the L method to execute directly corresponds to the number of points in the evaluation graph. Since the number of points in the evaluation graph is controlled by the number of clusters at the finest grain clustering, the L method runs much faster for clustering algorithms that do not have an overly-fine initial clustering. The L method is very general and contains no parameters or constants. The number of points along the x-axis of the evaluation graph is not a parameter. It is a result of the clustering algorithm used to generate those points. The maximum x value in the evaluation graph is either the number of clusters at the initial fine grain clustering in a bottom-up algorithm, or the number of clusters in the final clustering in a top-down algorithm.

3.3 Refinements to the L Method

Many bottom-up algorithms create an initial fine-grain clustering by initially treating every data point as a cluster. This can cause an evaluation graph to be as large as the original data set. In such an evaluation graph, the values representing merges at extremely fine-grain clusterings (large values of x) are irrelevant. Such a large number of irrelevant data points in the evaluation graph can prevent an "L" shaped curve.

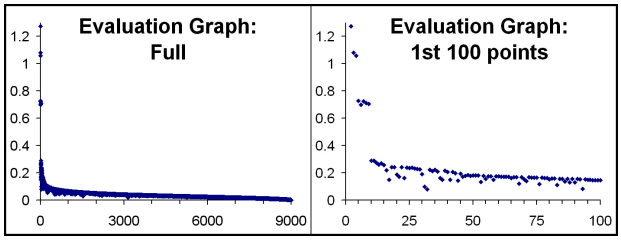

Figure 4. Full and partial evaluation graphs created by CURE. Only the first 100 points are shown on the right side.

Figure 4 shows a 9,000 point evaluation graph on the left, and the first 100 data points of the same graph on the right. The graph on the right is a more natural "L" shaped curve, and the L method is able to correctly identify that there are 9 clusters in the data set. However, in the full evaluation graph, there are so many data points to the right side of the "correct"

knee, that the very few points on the left of that knee become statistically irrelevant. The L method performs best when the sizes of the two lines on each side of the knee are reasonably balanced. When there are far too many points on the right side of the actual knee, the knee that is located by the L method will most likely be larger than the actual knee. In the full evaluation graph, containing 9,000 data points, the knee is incorrectly detected at x=359, rather than x=9. However, when many of the irrelevant points are removed from the evaluation graph, such as all points greater than x=100 (see the right side of Figure 4), the correct knee is located at x=9. The following algorithm iteratively refines the knee by adjusting the focus region and reapplying the L method (note that the clustering algorithm is NOT reapplied).

Iterative Refinement of the Knee

Input: *EvalGraph* (a full evaluation graph)
Output: the *x*-axis value location of the knee (also the suggested number of clusters to return)

```
 1| int cutoff =
 2|     lastKnee =
 3|     currentKnee = EvalGraph.size()
 4|
 5| REPEAT
 6| {
 7|   lastKnee = currentKnee
 8|   currentKnee = LMethod(evalGraph,cutoff)
 9|   cutoff = currentKnee*2
10| } UNTIL currentKnee ≥ lastKnee
11|
12| RETURN currentKnee
```

This algorithm initially runs the L method on the entire evaluation graph. The value of the knee becomes the middle of the next focus region and the L method becomes more accurate because the lines on each side of the true knee are becoming more balanced. Since the refinement stops when the knee does not move to the left after an iteration, the focus region decreases in size after every iteration. The true knee is located when the L method returns the same value as the previous iteration (line #10, or if the current pass returns a knee that has a roughly balanced number of points on each side of the knee (also line #10). The 9,000 point evaluation graph in Figure 4 takes four iterations to correctly determine that there are 9 clusters in the data set (9,000→359→15→9→9). The cutoff value is not permitted to drop below ~20 in the "LMethod()," because a reasonable number of points are needed for the two fitted-lines to fit actual trends, rather than detecting spurious trends indicated by a small number of points in the evaluation graph. The minimum cutoff size of 20 performed well on all tests that have been run to date and it will most likely never need to be changed. The minimum cutoff size can therefore most likely be treated as a constant rather than a parameter (keeping the L method 'parameterless').

Iteratively refining the knee does not significantly increase the execution time of the L method. Iterative refinement converges on the knee in very few iterations (usually less than three), and the first iteration is run with an evaluation graph that is much larger than those in later iterations. The L method is an $O(N^2)$ algorithm with respect to the size of the evaluation graph. Evaluation graphs with fewer than 1,000 points can be evaluated in less than a few seconds; however, a 9,000 point evaluation graph takes several minutes. In practice, it is usually permissible to ignore points in an evaluation graph past some large number when it is unlikely or undesirable to have such a large number of clusters.

Another potential problem is that sometimes the evaluation graph will reach a maximum (moving from right to left) and then start to decrease. This can be seen in Figure 2, where the distance between the closest segments reaches a maximum at x=4. This can prevent an "L" shaped curve from existing in the evaluation graph. The data points to the left of the maximum value (the 'worst' merge) can be ignored. This occurs in some algorithms that have distance functions that become undefined when the remaining clusters are extremely dissimilar to each other.

4. EMPIRICAL EVALUATION

The goal of this evaluation is to demonstrate the ability of the L method to identify a reasonable number of clusters to return in hierarchical clustering and hierarchical segmentation algorithms. Each algorithm will be run on a number of data sets and the number of clusters that the L method identifies is compared to the 'correct' answer. Existing methods to determine the number of segments or clusters in a data set will also be evaluated on the same data sets, and their performance will be compared to that of our L method.

4.1 Identifying the Number of Clusters

4.1.1 Procedures and Criteria

The seven diverse data sets used to evaluate the L method for clustering algorithms vary in size, number of clusters, separation of clusters, density, and amount of outliers. There are some data sets that contain only spherical clusters, and some which contain very non-spherical clusters, including clusters within clusters. The seven spatial data sets that were used are (see Figure 5):

1. A data set with four well separated spherical clusters (4,000 pts).
2. Nine square clusters connected at the corners (9,000 pts).
3. Ten spherical clusters. Five overlapping clusters similar to data set #7 (not shown), as well as five additional well separated clusters and a uniform distribution of outliers (5,200 pts).
4. Ten well separated clusters of varying size and density (5,000 pts).

5. A 9 cluster data set used in the Chameleon paper, but with the outliers removed. Non-spherical clusters with clusters completely contained within other clusters (~9,100 pts).
6. An 8 cluster data set containing non-spherical clusters with clusters partially enveloping other clusters (~7,600 pts).
7. Five spherical clusters of equal size and density. The clusters are slightly overlapping (5,000 pts, not in Figure 5).

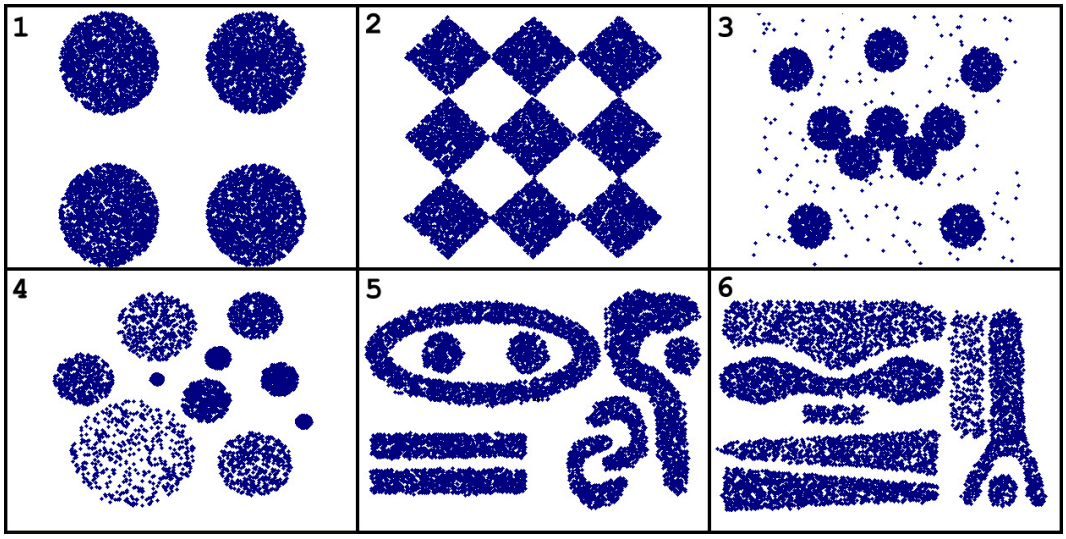

Figure 5. Data sets 1-6 for evaluating the L method in clustering algorithms (data set #7 not shown).

The clustering algorithms used to test the L method were Chameleon and CURE. Chameleon was implemented locally and was run with the parameters: k=10 (k nearest neighbors, not k clusters), *minSize*=3%, and α=2. CURE was implemented as specified in the CURE paper [6], with the shrinking factor set to 1/3 and the number of representative points for each cluster set to 10.

CURE, Chameleon, and *K*-means was used to evaluate the Gap Statistic's relative performance against that of the L method. The Gap Statistic was calculated using the Gap/unf variant [21]. The Gap statistic must be run for each user-specified potential value for the number of clusters. The potential number of clusters evaluated by the Gap statistic were k={2…20}. Both the L method and the Gap statistic were tested on CURE and Chameleon for a direct comparison. However, the Gap statistic was also evaluated on the *K*-means algorithm, because *K*-means was used by Tibshirani, Walther, and Hastie to evaluate the Gap statistic [21]. It is important to note that for every different clustering algorithm, the L method's evaluation graph contains values created by the particular clustering algorithm's evaluation function, while the Gap statistic uses pairwise distances to evaluate clusters regardless of the clustering algorithm that produced them. Thus, the L method's performance is more likely to be consistent over different clustering algorithms, while the Gap statistic will only work well for clustering algorithms that measure cluster quality similar to its own fixed method.

The experimental procedure for evaluating the performance of the L method for hierarchical clustering algorithms consists of running the CURE and Chameleon clustering algorithms, which have been modified to determine the number of clusters to return through use of the L method, on seven diverse data sets (shown in Figure 5). The number of clusters determined will be compared to the correct number of clusters. The data sets are synthetic, so the correct number of clusters is known. These results will also be compared to the number of clusters suggested by the existing Gap statistic for CURE, Chameleon, and also the *K*-means clustering algorithm.

4.1.2 Results and Analysis

The correct number of clusters was determined by the L method 6 out of 7 times for Chameleon and 4 out of 5 times for the CURE algorithm. The results are contained in Table 1. The actual number of clusters suggested for CURE on data set #3 was 9. However, in the presence of outliers, CURE creates a number of very small clusters that contain only outliers. After removing these small clusters, only 6 true clusters remained. Data sets #5 and #6 contain complex clusters and could only be properly clustered by Chameleon; "N/A" is placed in the cells of Table 1 where the number of clusters suggested was not evaluated because the clustering algorithm was unable to produce the correct set of clusters.

Table 1. Results of using the L method and the Gap statistic with various clustering algorithms.

Data Set	**Data set #**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	*Exact Matches*
	Correct # of clusters	4	9	10	10	9	8	5	
Num of clusters predicted by L Method	*Chameleon*	4	9	11	10	9	8	5	6 of 7
	CURE	4	9	6 (9)	10	N/A	N/A	5	4 of 5
Num of clusters predicted by Gap Statistic	*Chameleon*	4	2	2	2	2	2	2	1 of 7
	CURE	4	2	2	2	N/A	N/A	2	1 of 5
	K-means	4	2	2	2	N/A	N/A	2	1 of 5

The Gap statistic was only able to determine the correct number of clusters for one of the seven data sets, regardless of the clustering algorithm used. The Gap statistic performs similarly to many existing methods [21], and only works well for well-separated, circular clusters (only data set #1 satisfies these constraints). The Gap statistic tended to suggest far too few clusters because the cluster separation was not great enough for it to consider the clusters to be distinct.

The correct number of clusters was not determined by either algorithm for data set #3, which contains many outliers and a mixture of both well separated and overlapping clusters. In the evaluation graph for CURE, there is a large smooth knee that spans approximately 200 data points. Most merges in this region are between outliers, but there are also merges of the five overlapping clusters mixed in. There is no sharp knee until after all of the five overlapping clusters have already been

merged together. The six clusters returned by the L method were the five well separated clusters, and the group of overlapping clusters in the center (see data set #3 in Figure 5). Even though the L method recommends four fewer clusters than the 'correct' answer, the six recommended clusters have a more uniform separation than the 'correct' answer. In this case, the best number of clusters is open to interpretation. The answer given for Chameleon on data set #3 was off by one because the knee of the curve was not sharp enough for the L method to identify the exact number of clusters. This is most likely due to a weakness in our Chameleon implementation, which does not contain a graph bisection algorithm that is as powerful as the one described in the Chameleon [10] paper.

The L method determines the number of clusters to return by examining the evaluation graphs produced by each clustering algorithm. Examples of evaluation graphs are shown in Figure 6, where the *x*-axis is the number of clusters, and the *y*-axis is the value of the clustering algorithm's evaluation function at *x* clusters. Notice that the *y*-axis values in CURE evaluation graphs increase from right to left, while the Chameleon evaluation graphs decrease from right to left. This is because CURE's evaluation metric measures distance and Chameleon's evaluation metric measures similarity.

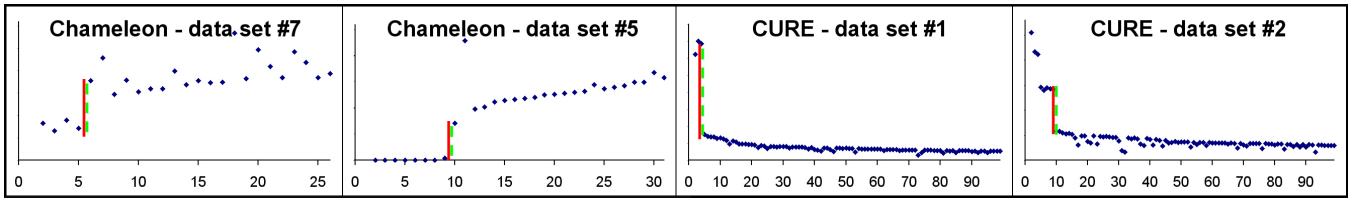

Figure 6. Actual number of clusters and the correct number predicted by the L method (*axes*: *x*= # of clusters, *y*=evaluation metric – *lines*: *solid lines*=correct # of clusters, *dashed lines*=# of clusters determined by L method).

In Figure 6, the solid line indicates the correct number of clusters, while the dashed line indicates the number of clusters suggested by the L method. The lines are right next to each other in each case which indicates that the correct number of clusters was determined. The best number of clusters is usually just before a large jump in the evaluation graph. To the left of the jump dissimilar clusters have been merged together creating inhomogeneous clusters; and to the right of the jump there are too many clusters that are similar to each other. Since a 'good' cluster is both internally homogeneous and dissimilar to other clusters, the location of the jump should be a good measure of the best number of clusters.

The L method runs more quickly for clustering algorithms that do not have an overly-fine initial clustering because these algorithms create smaller evaluation graphs. Chameleon initially produces fine grain clusterings that contain fewer than 100 clusters and the L method needs less than 0.01 seconds to determine the number of clusters. CURE produces the finest initial clustering possible, which creates evaluation graphs with up to 8,999 points in our evaluation. With CURE, the L method's run-time is between 40 seconds and 4 minutes. This execution time can be drastically reduced by only evaluating

the first *maxk* points in the evaluation graph, where *maxk* is some large number guaranteed to be more than the actual number of clusters. In our evaluation, the L method determined the number of clusters to return in less than 0.9% of the total execution time for CURE and less than 0.003% of the total execution time for Chameleon. Runtime for the Gap statistic is significantly slower. The Gap statistic must be run for each potential number of clusters, and since it calculates pairwise distances within a cluster, its run-time (to evaluate just a single potential number of clusters) approaches $O(N^2)$. In our evaluation, the Gap statistic took up to 28 minutes to evaluate the clusterings in the range $k=\{2\ldots20\}$, and took several times longer to evaluate each clustering than the clustering algorithm needed to produce it.

4.2 Identifying the Number of Segments

4.2.1 Procedures and Criteria

The experimental procedure for evaluating the L method in segmentation algorithms consists of running two different segmentation algorithms on seven different data sets and determining if a 'reasonable' number of segments is suggested by the L method. This number of segments suggested will then be compared to the 'correct' number of segments, and also the number suggested by the existing permutation tests method [22].

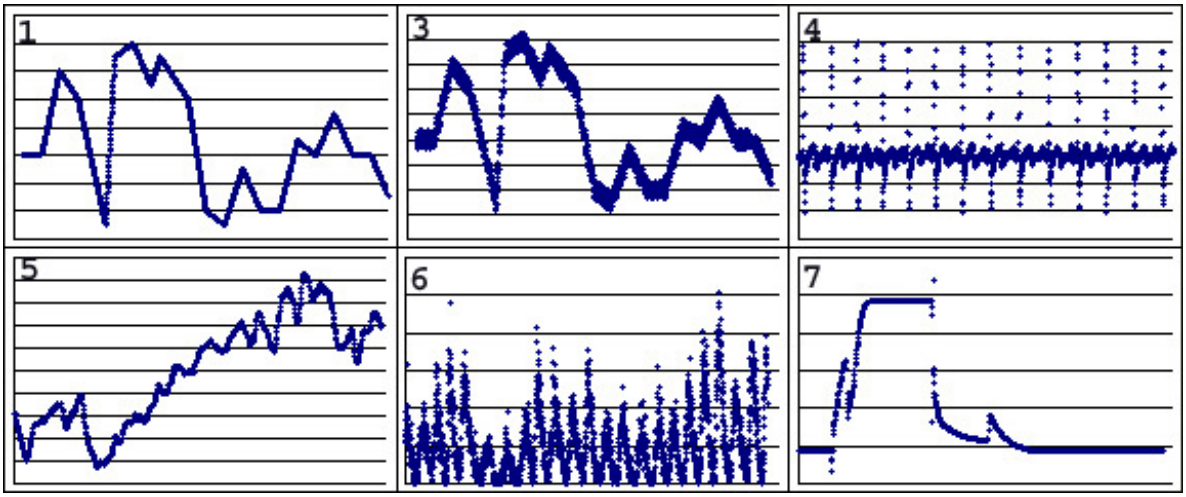

Figure 7. Data sets 1, 3, 4, 5, 6, and 7 for evaluating the L method in segmentation algorithms.

The time series data sets used to evaluate the L method for hierarchical segmentation algorithms are a combination of both real and synthetic data. The seven time series data sets used for this evaluation (shown in Figure 7) are:

1. A synthetic data set consisting of 20 straight line segments (2,000 pts).
2. The same as #1, but with a moderate amount of random noise added (2,000 pts, not in Figure 7).
3. The same as #1, but with a substantial amount of random noise added (2,000 pts).

4. An ECG of a pregnant woman from the Time Series Data Mining Archive [12]. It contains a recurring pattern (a heart beat) that is repeated 13 times (2,500 pts).
5. Measurements from a sensor in an industrial dryer (from the Time Series Data Mining Archive [12]). The time series appears similar to random walk data (876 pts).
6. A data set depicting sunspot activity over time (from the Time Series Data Mining Archive [12]). This time series contains 22 roughly evenly spaced sunspot cycles, however the intensity of each cycle can vary significantly (2,900 pts).
7. A time series of a space shuttle valve energizing and de-energizing (1,000 pts).

A 'correct' number of segments for a data set and segmentation algorithm is obtained by running the algorithm with various values of *k* (controls the number of segments returned), and determining what particular value(s) or range of values of *k* produces a 'reasonable' PLA. The PLAs that are considered 'reasonable' are those at a value of *k*, where no adjacent segments are very similar to each other and all segments are internally homogeneous (segments have small error). The synthetic data sets have a single correct value for *k*. The real sets have no single correct answer, but rather a range of reasonable values. The reasonable and best numbers of segments for the real data sets may vary for each algorithm. A single 'best' number of segments cannot be used for all of the segmentation algorithms because one number that produces the best set of segments for one algorithm may produce a poor set of segments for another on the same data set.

The segmentation algorithms used in this evaluation were Gecko [16] and bottom-up segmentation (BUS). BUS (bottom-up segmentation) is a hierarchical algorithm that initially creates many small segments and repeatedly joins adjacent segments together. More specifically, BUS evaluates every pair of adjacent segments and merges the pair that causes the smallest increase in error when they are merged together. BUS was tested with the L method using two different values on the *y*-axis of the evaluation graph. The two variants are named BUS-greedy and BUS-global. BUS-greedy's *y*-axis in the evaluation graph is the increase in error of the two most similar segments when they are merged, and BUS-global's *y*-axis is the error of the entire linear approximation when there are *x* segments (absolute error). The existing 'permutation tests' method was also evaluated using BUS.

Both Gecko and BUS made use of an initial top-down pass to create the initial fine-grain segments. The minimum size of each initial segment generated in the top down pass was 10. For the permutation test algorithm, *p* was set to 0.05, and 1,000 permutations were created. The parameter *p* controls the percentage of permutated time series that must be increasing in quality faster than the original time series to stop creating more segments.

4.2.2 Results and Analysis

A summary of the results of the L method's and permutation tests' ability to determine the number of segments to return from segmentation algorithms is contained in Table 2. For both Gecko and BUS, the 'reasonable' range of correct

answers is listed. These ranges may vary between the two algorithms because BUS and Gecko do not merge segments in exactly the same sequence. However, BUS-greedy, BUS-global, and permutation tests all produce identical PLAs for *k* segments, and therefore have identical 'reasonable' answers. The first three data sets are synthetic and have a single correct answer, but the other data sets have a range of 'reasonable' answers. Data set #5 is similar to random walk data, and any number of segments seemed reasonable because there was no underlying structure in the time series.

		Data Set	**1**	**2**	**3**	**4**	**5**	**6**	**7**	*Reasonable Range Matches*
Gecko		*Reasonable # of segments*	*20*	*20*	*20*	*42-123*	*?*	*44-57*	*9-20*	
	Gecko w/L Method	*Num of segments Given*	20	20	N/A	92	32	45	17	5 of 5
Bottom-up Segmentation		*Reasonable # of segments*	*20*	*20*	*20*	*42-123*	*?*	*45-53*	*14-21*	
	BUS-greedy w/ L method	*Num of segments Given*	20	20	20	46	14	48	9	5 of 6
	BUS-global w/ L method	*Num of segments Given*	20	20	19	106	39	39	13	3 of 6
	BUS w/ permutation Tests	*Num of segments Given*	25	34	25	2	15	6	65	0 of 6

Table 2. Results of using the L method with three hierarchical segmentation algorithms.

The L method worked very well for both BUS-greedy and Gecko. It correctly identified a number of segments for BUS-greedy that was within the reasonable range in 5 out of the 6 applicable data sets. Gecko, which also uses a greedy evaluation metric (but uses slope rather than segment error), had the L method suggest a number of segments within the reasonable range for all 5 applicable data sets. Gecko was unable to correctly segment data set #3 (indicated by "N/A" in Table 2) because it contained too much noise. In all but one test case (10 of 11), the L method was able to correctly determine that the three synthetic data sets contained exactly twenty segments. BUS-global did not perform quite as well. The L method was only able to return a reasonable number of segments for BUS-global in half of its test cases, but all of its incorrect answers were close to being correct.

Permutation tests did not perform well and never determined a reasonable number of segments. The reason that permutation tests did poorly varied depending on the data set. Data set #1 is synthetic and contains no noise, which allows a PLA to approximate it with virtually zero error. However, measuring a relative increase in error when the error is near zero causes unexpected results because relative increases are either very large or undefined when the error is at or near zero. For data set #4 and #6, the relative change in approximation error is rather constant regardless of the number of segments. On

data set #4, the PLA between 2 and 3 segments has nearly zero relative change in error, which causes permutation tests to incorrectly assume that the data has been over-fitted and stop producing segments prematurely. An example of far too many segments being returned occurs on data set #7, where the relative error of the time series never falls below the relative error of the permutations until far too many segments are produced.

Some of the evaluation graphs used by the L method for Gecko, BUS-greedy, and BUS-global are shown in Figure 8. The third evaluation graph in Figure 8 contains the L method's evaluation graph for Gecko on data set #1, the noise-free synthetic data set. The *x*-axis is the number of segments, and the *y*-axis is Gecko's evaluation metric (distance between two closest adjacent segments) at *x* segments. The evaluation graph is created right to left as segments are meged together. In this case, the correct number of segments is easily determined by the L method because there is a very large jump at *x*=20. In evaluation graph on the right side of Figure 8 (data set #7 BUS-global), the range of reasonable answers lies between the two long lines. The range is larger than for data set #1 because the segments have less 'separation' and there is no sharp knee. Instead there is a range of good answers. However, the L method suggests a number of segmetns that just misses the reasonable range.

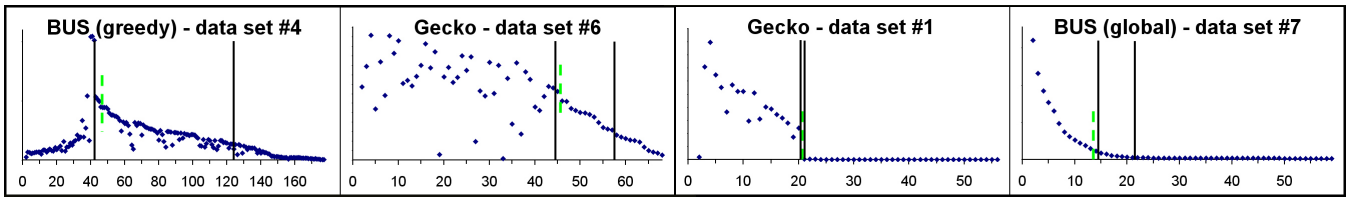

Figure 8. The reasonable range for the number of segments and the number returned by the L method. (*axes*: *x*=# of segments, *y*=evaluation metric – *short dashed line*=# of segments determined by the L method, *long solid lines*=marks the boundaries of the reasonable range for the # of segments.

In the evaluation graph at the left of Figure 8 (data set #4 BUS-greedy), the L method returned a number of segments that was towards the low end of the reasonable range. Remember, that for segmentation algorithms, all data ponits to the left of the data point with the maximum value are ignored (discussed in the last section of 3.3). The best number of segments is 42. At 42 segments each heart beat contains approximately 3 segments. If there are fewer than 42 segments, they are no longer homogeneous. However, PLAs with significantly more segments (up to 123) are still reasonable because each new segment still significantly reduces the error. However, if there are more than approximately 123 segments, adjacent segments start to become too similar to each other.

The second evaluation graph shown in Figure 8 (data set #6 Gecko) also has 'better' PLAs when the number of segments is near the low end of the reasonable range (fewer segmetns). This is common because the best set of segments is

often the minimal set of segments that adequately represents the data. Even though there is apparently no significant knee in this evaluation graph, a good number of segments can still be found by the L method. This is because the knee found by the L method does not necessarily have to be the point of maxium curvature. It may also be the location between the two regions that have relatively steady trends. Thus, the L method is able to determine the location where there is a significant change in the evaluation graph and it becomes erratic (x<44). In this case it indicates that too many segments have been merged together and the distance function is no longer as well-defined.

The poorer performance of BUS-global (compared to Gecko and BUS-greedy) is due to a lack of prominence in the knee of the curve compared to greedy methods (see lower-right graph in Figure 8). Greedy evaluation metrics increase more sharply at the knee, while global metrics have larger more ambiguous knees in their evaluation graph. A potential problem is if more than one knee exists in the evaluation graph. This is typically not a problem if one knee is significantly more prominent than the others. If there are two equally prominent knees, the L method is likely to return a number of segments that falls somewhere between those two knees. This is acceptable if all of the values between the two knees are reasonable. If not, a poor number of segments will most likely be returned by the L method.

The L method took approximately 0.01 seconds to determine the number of segments in every test cases and the segmentation algorithms took anywhere from 9 to 30 seconds to execute. The L method never required more than 0.1% of the total execution time to determine the number of segments. In stark contrast, permutation tests required up to 5 hours because each permutation of the original time series had to be segmented.

5. CONCLUDING REMARKS

We have detailed our L method, which has been shown to work reasonably well in determining the number of clusters or segments for hierarchical clustering/segmentation algorithms. Hierarchical algorithms that have greedy evaluation metrics perform especially well. In our evaluation, the L method was able to determine a reasonable number of segments in 10 out of 11 instances for greedy hierarchical segmentation algorithms, and a correct number of clusters in 10 of 12 instances for hierarchical clustering algorithms. Algorithms with global evaluation metrics did not work as well with the L method because the knees in the evaluation graphs are not as prominent and easy to detect. The Gap statistic and permuation tests were also evaluated and the L method achieved much better results in our evaluation. The L method is also much more efficient than both the Gap statistic and permutation tests, typically requiring only a fraction of a second to determine the number of clusters rather than minutes or even many hours in the case of permutation tests.

Iterative refinement of the knee is a very important part of the L method. Without it, the L method would only be effective in determining the number of clusters/segments within a certain range. The iterative refinement algorithm explained in this paper enables the L method to always run under optimal conditions: balanced lines on each side of the knee no matter how large the evaluation graph is or where the knee is located.

Like most existing methods, the L method is unable to determine if the entire data set is an even distribution and consists of only a single cluster (the null hypothesis). However, the L method also has the limitation that it cannot determine if only two clusters should be returned.

Future work will involve testing the L method with additional clustering and segmentation algorithms on additional data sets to help further understand the algorithm's strengths and weaknesses. We will also explore modifications to the L method that will enable it to determine when only one or two clusters should be returned. Work will also focus on comparing the L method to additional methods that attempt to determine the number of clusters in a data set.

6. REFERENCES

[1] Baxter, R. A. & J. J. Oliver. The Kindest Cut: Minimum Message Length Segmentation. In *Algorithmic Learning Theory, 7th Intl. Workshop*, 83-90. Sydney, Australia, 1996.

[2] Chiu, T., D. Fang, J. Chen, Y. Wang & C. Jeris. A Robust and Scalable Clustering Algorithm for Mixed Type Attributes in Large Database Environment. In *Proc. Of the 7th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pp. 263-268, 2001.

[3] Ester M., Kriegel H., Sander J., and Xu X. (1996) A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining*, AAAI Press.

[4] Foss, A. & A. Zaïane. A Parameterless Method for Efficiently Discovering Clusters of Arbitrary Shape in Large Datasets. In *IEEE Intl. Conf. on Data Mining*, 2002.

[5] Fraley, C. & E. Raftery. How many clusters? Which clustering method? Answers via model-based Cluster Analysis. In *Computer Journal*, vol. 41, pp. 578-588, 1998.

[6] Guha, S., R. Rastogi & K. Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proc. Of ACM SIGMOD Intl. Conf. on Management of Data*, pp. 73-82, 1998.

[7] Harris, S., D. R. Hess & J. Venegas. An Objective Analysis of the Pressure-Volume Curve in the Acute Respiratory Distress Syndrome. In *American Journal of Respiratory and Critical Care Medicine*, vol. 161, number 2, pp. 432-439, 2000.

[8] Hansen, M. & B. Yu. Model Selection and the Principle of Minimum Description Length. In *JASA*, vol. 96, pp.746-774, 2001.

[9] Hinneburg A., Keim D. (1998) An Efficient Approach to Clustering in Large Multimedia Databases with Noise. Proc AAAI.

[10] Karypis, G., E. Han & V. Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*, 32(8) pp. 68-75, 1999.

[11] Keogh, E., S. Chu, D. Hart & M. Pazanni. An Online Algorithm for Segmenting Time Series. In *Proc. IEEE Intl. Conf. on Data Mining*, pp. 289-296, 2001.

[12] Keogh, E. & T. Folias (2003). The UCR Time Series Data Mining Archive [http://www.cs.ucr.edu/~eamonn/TSDMA/ index.html]. Riverside, CA. University of California – Computer Science and Engineering Department.

[13] Monti, S., T. Pablo, J. Mesirov & T. Golub. Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data. In *Machine Learning*, 52, pp. 91-118, 2003.

[14] Ng R., Han J. (1994) Efficient and effective clustering method for spatial data mining. In Proc. Conf. on Very Large Data Bases, pp 144-155.

[15] Roth, V., T. Lange, M. Braun & J. Buhmann. A Resampling Approach to Cluster Validation. In *Intl. Conf. on Computational Statistics*, pp. 123-129, 2002.

[16] Salvador, S., P. Chan & J. Brodie. Learning States and Rules for Time Series Anomaly Detection. In *Proc. of the 17th Intl. Flairs Conference*, 2004.

[17] Seikholeslami G., Chatterjee S., and Zhang A. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. Proceedings of the 24th VLDB Conference, 1998

[18] Smyth, P. Clustering Using Monte-Carlo Cross-Validation. In *Proc. 2nd KDD*, pp.126-133, 1996.

[19] Sugiyama, M. & H. Ogawa. Subspace Information Criterion for Model Selection. In *Neural Computation*, vol. 13, no.8, pp. 1863-1889, 2001.

[20] Tibshirani, R., G. Walther, D. Botstein & P. Brown. Cluster Validation by Prediction Strength, Technical Report, 2001-21, Dept. of Biostatistics, Stanford Univ, 2001

[21] Tibshirani, R., G. Walther & T. Hastie. Estimating the number of clusters in a dataset via the Gap statistic. In *JRSSB*, 2003.

[22] Vasko, K. & T. Toivonen. Estimating the number of segments in time series data using permutation tests. In *Proc. IEEE Intl. Conf. on Data Mining*, pp. 466-473, 2002.

[23] Zhang, T., R. Ramakrishnan & M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases, In ACM *SIGMOD Intl. Conf. on Management of Data and Symposiium on Principles of Database Systems*, pp. 103-114, 1996.