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

Stan Salvador and Philip Chan
Dept. of Computer Sciences Technical Report CS-2003-18
Florida Institute of Technology
Melbourne, FL 32901
{ssalvado, pkc}@cs.fit.edu

ABSTRACT

We investigate techniques to automatically determine the number of clusters 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.

1. INTRODUCTION

Motivation. 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 automatically find a reasonable number of clusters, many existing methods must be run repeatedly with different parameters (trial and error), and are impractical for real world data sets, which can be quite large.

Problem. We desire to develop an algorithm that can automatically and efficiently determine a reasonable number of clusters/segments to return from any hierarchical clustering/segmentation algorithm.

Approach. In order to identify the correct number of clusters to return from a hierarchical clustering/segmentation algorithm, we introduce the L Method. Hierarchical algorithms either merge the two most similar clusters together (bottom-up), or split the least internally homogeneous cluster into two (top-down). The definition of a "cluster" is not well-defined and measuring cluster quality is rather subjective. Thus, there is a large number of clustering algorithms with unique similarity functions and correspondingly unique notions of what a good cluster "looks" like. The L Method uses the same similarity function used in a hierarchical algorithm to construct an evaluation graph where the *x*-axis is the number of clusters and the *y*-axis is the value of the similarity function during the merge or split at *x* clusters. This 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. Our approach needs only a single pass over the dataset, and the overhead is trivial compared to the runtime of the clustering/segmentation algorithm.

Contributions. Our main contributions are: (1) we demonstrate an accurate 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 and limitations 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.

Organization. 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 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, hierarchical, density-based, and grid-based. Hierarchical clustering algorithms can be 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 [3] and Chameleon [5] are examples of two hierarchical clustering algorithms.

Segmentation Algorithms. Segmentation algorithms take time series data as input and produce a Piecewise Linear Representation (PLR) as output. A PLR is a set of consecutive line segments that tightly fit the original time series. Segmentation algorithms are somewhat 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 common approaches to time series segmentation [6]. 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, the entire time series is recursively 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 2 most similar adjacent segments are repeatedly joined until the desired number of segments or the error threshold is reached. Gecko [9] is a bottom-up segmentation algorithm that merges clusters based on slope and creates its initial fine-grain approximation by first performing a top-down pass.

Determining the Number of Clusters/Segments. Six 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 fid the data as accurately as possible. Monte Carlo cross-validation [10] 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 attempts to minimize the complexity of the model. Specific methods to penalize models based on their complexity are: MML [1], MDL [4], BIC [2], AIC, and SIC [11]. Permutation tests [14] have been used to prevent segmentation algorithms from creating a PLR that over-fits the data. Resampling [8] attempts to find the correct number of clusters by repeatedly clustering samples of the data set, and determining at what number of clusters the clusterings of the various samples are the most "stable."

The "knee" of an error curve is a well known, but little understood method to determine an appropriate number of clusters or segments. There are other methods that transform the error curve into a new curve where the minimum or maximum value in the transformed curve is used as the number of clusters/segments to return. Such methods include the Gap statistic [13] and prediction strength [12]. These methods generally require the entire clustering or segmentation algorithm to be run for each potential value of *k*.

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. classification error" 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 first data point with a second derivative above some threshold value.
3. The data point with the largest second derivative.
4. The point on the curve that is furthest from a line fitted to the entire curve.
5. 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. The third method uses more than one pair of points, but still only considers 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 4th 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 automatically determine an appropriate number of clusters/segments to return, by finding the knee in an evaluation graph.

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. Other approaches use similar graphs, however they are usually generated by re-running the entire clustering or segmentation algorithm for every value on the *x*-axis, which is quite inefficient. In hierarchical algorithms, which either split or merge clusters at each step, all clusterings containing '*1*' to '*the* number *of clusters in the initial (or final) 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. A common example is the sum of all the pairwise distances between points in each cluster. Most global evaluation metrics are computed in $O(N^2)$ time. Thus, in many cases, it takes longer to evaluate a single set of clusters than it takes to create them. Since the evaluation function must be run for every potential number of clusters, this method is often too inefficient to be practical. The alternative is to use greedy measurements. The greedy method works in hierarchical algorithms by evaluating only the two clusters that are involved in the current merge or split, rather than the entire data set.

Many "external" evaluation methods attempt to determine a reasonable number of clusters by evaluating the output of an arbitrary clustering algorithm. Each evaluation method has its own notion of cluster similarity. Most external methods use distance functions that are heavily biased towards spherical clusters. Such methods would be unsuitable for a clustering algorithm that has a different notion of cluster distance/similarity. For example, Chameleon [5] uses a complex similarity function that can produce interesting non-spherical clusters, and even clusters within clusters. Therefore, the L Method is integrated into the clustering algorithm and the metric used in the evaluation graph is the same metric used in the clustering algorithm.

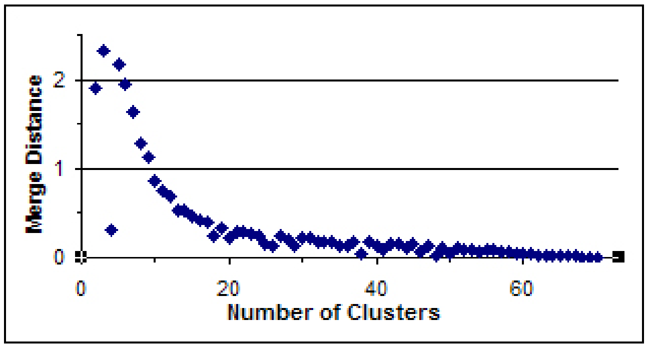

Figure 1. A sample evaluation graph.

An example of an evaluation graph produced by the Gecko segmentation algorithm 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 closest clusters are used to generate the value on the *y*-axis. The curve in Figure 1 has three distinctive areas: a rather 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 end, 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 clustering is becoming poor because clusters are no longer internally homogeneous. If the best available remaining merges start becoming increasingly poor, it means that too many merges have already been performed.

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 region contain a balance of clusters that are both highly homogeneous, and also dissimilar to each other. To the left of the knee clusters are no longer homogeneous, and to the right of the knee clusters are too similar to each other. Determining the number of clusters where this region exists will therefore give a reasonable number of clusters to return.

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.

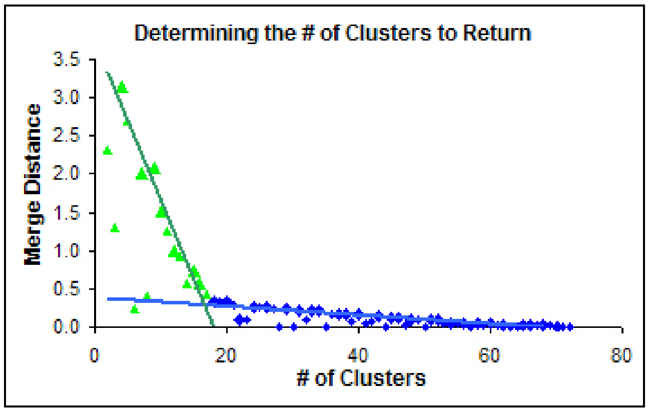

Figure 2. Finding the number of clusters using the L Method.

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 pair that fits their respective data points with the minimum amount of error is the pair on the bottom left.

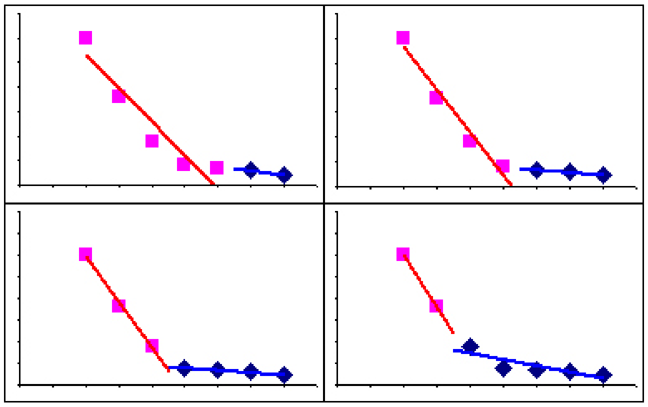

Figure 3. All four possible pairs of best-fit lines for a small evaluation graph.

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…*b*, where *c*=3…*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*:

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

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, $\hat{c}$, such that $RMSE_c$ is minimized, that is:

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

The location of the knee at $x=\hat{c}$ is used as the number of clusters to return.

3.3 Refinements to the L Method

Iterative Refinement of the Knee. Some 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. If such an evaluation graph has thousands of merge values, the ones 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, or more specifically a flat region to the right of the knee.

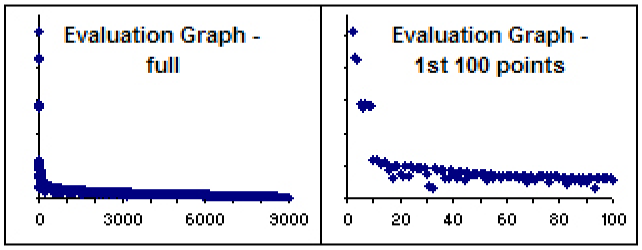

Figure 4. Full and partial evaluation graphs created by CURE. Only the first 100 points are shown on the right side.

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 iteration stops when the knee does not move to the left, the focus region decreases in size. 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. The cutoff value is not permitted to drop below 20 in the "LMethod()," because the L Method does not work well if only a very small number of points are in the evaluation graph.

Refinements for Segmentation Algorithms. Evaluation graphs for some segmentation algorithms can often be very jumpy when segmenting noisy data. The exact nature of the curve may be easily to determine visually, but there can be many points that do not fit the curve. These stray points generally do not occur consecutively. These stray points can prevent the L Method from accurately locating the knee. However, because they do not usually occur consecutively, the curve can be smoothed by only using the highest valued point of every consecutive pair when computing the best-fit lines of the curve.

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 clusters 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 because they represent clusterings that have already had very dissimilar segments merged together.

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.

4.1 Identifying the Number of Clusters

4.1.1 Procedures and Criteria

The seven diverse data sets used to evaluate the L Method in clustering algorithms vary in size, number of clusters, separation of clusters, density, and amount of outliers.

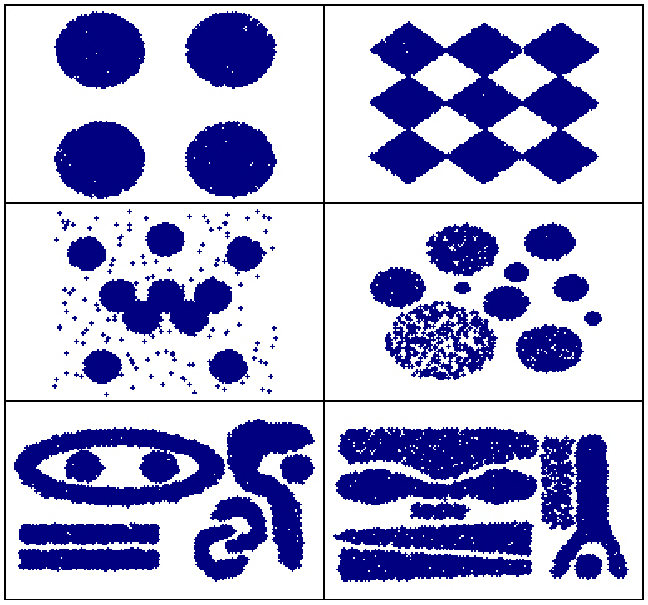

Figure 5. Data sets 1, 2, 4, 5, 6, and 7 for evaluating the L Method in clustering algorithms.

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. Five spherical clusters of equal size and density. The clusters are all close to each other and slightly overlapping (5,000 pts, not in Figure 5).
4. Ten spherical clusters. Five overlapping clusters similar to data set #3, as well as five additional well-separated clusters and a uniform distribution of outliers (5,200 pts).
5. Ten well-separated clusters of varying size and density (5,000 pts).
6. 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 and a moderate amount of outliers (~9,100 pts).
7. An 8 cluster data set used in the Chameleon paper, but with the outliers removed. Non-spherical clusters with clusters partially enveloping other clusters and a moderate amount of outliers (~7,600 pts).

The clustering algorithms used are Chameleon and CURE. Chameleon was implemented locally and was run with the parameters: k=10, *minSize*=3%, and α=2. CURE was implemented as specified in the CURE paper [3], with the shrinking factor set to 1/3 and the number of representative points for each cluster set to 10.

The experimental procedure for evaluating the performance of the L Method for hierarchical clustering algorithms consists of running all four clustering algorithms, which have been modified to automatically 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 automatically returned by the clustering algorithm will be compared to the correct number of clusters.

4.1.2 Results and Analysis

The correct number of clusters was determined 6 out of 7 times for Chameleon and 4 out of 5 times for the CURE algorithm. A summary of the results is contained in Table 1. The actual number of clusters suggested for CURE on data set #4 was 9. In the presence of outliers, CURE creates many very small clusters that contain only outliers. After removing these small clusters, only 6 clusters remained. Data sets #6 and #7 contain complex clusters and could only be properly clustered by Chameleon.

Table 1. Results of using the L Method with two hierarchical clustering algorithms.

Data Set		**Number of Clusters Predicted**	
data set #	*correct number of clusters*	*Chameleon*	*CURE*
1	4	4	4
2	9	9	9
3	5	5	5
4	10	11	6 (9)
5	10	10	10
6	9	9	N/A
7	8	8	N/A

Examples of evaluation graphs produced by the clustering algorithms are shown in Figure 6. Notice that the y-axis values in CURE evaluation graphs generally increase from right to left, and the Chameleon evaluation graphs generally decrease from right to left. This is because CURE’s evaluation metric is cluster distance and Chameleon’s evaluation metric is cluster similarity.

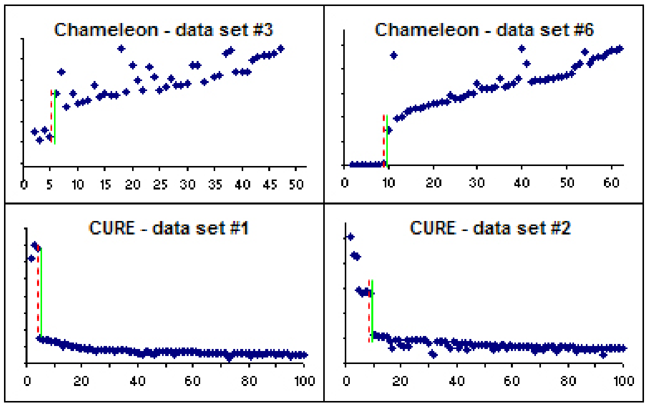

Figure 6. Actual number of clusters and the correct number predicted by the L Method (*<u>axes</u>*: *x*= # of clusters, *y*=evaluation metric – *<u>lines</u>*: *solid lines*=correct # of clusters, *dashed lines*=# of clusters determined by L Method).

The correct number of clusters was not determined for either algorithm on data set #4, which contained many outliers. The poor performance in the presence of outliers is due to a lack of a knee at a position that would indicate the correct number of clusters. In the evaluation graph for CURE, there is a large smooth knee that spans approximately 200 data points. Most of these merges in this region are between outliers, but there are 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 clusters returned by CURE were not 'correct', but they weren't too bad either. The six clusters returned were the five well-separated clusters, and the connected clusters in the center were all treated as a single cluster. A correct answer probably would have been determined by the L Method if the center cluster of 5 clusters were well separated. The answer given for Chameleon was only off by one because the knee was also off by one. This may be due to a weakness in our Chameleon implementation, which does not use a graph bisection algorithm that is as powerful as the one used in the original Chameleon implementation.

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 three different segmentation algorithms on seven different data sets and determining if a 'reasonable' number of segments is suggested by the L Method.

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

1. A synthetic data set consisting of 20 perfectly 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 [7]). 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 [7]). The time series seems to have a fractal structure (876 pts).
6. A data set depicting sunspot activity over time (from the Time Series Data Mining Archive [7]). 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).

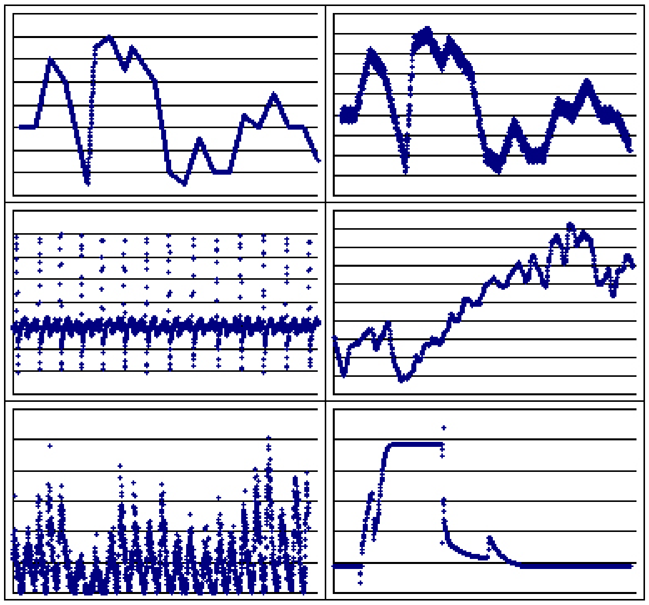

Figure 7. Data sets 1, 3, 4, 5, 6, 7 for evaluating the L Method in segmentation algorithms.

The 'correct' number of segments for a data set and a particular 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' PLR (piecewise linear representation). The PLRs 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 (roughly linear). The synthetic data sets have a single correct value for *k*. The other data sets have no single correct answer, but rather a range of reasonable values. If there are 'best' values of *k* within the reasonable range, they are recorded.

The segmentation algorithms used in this evaluation are: Gecko, bottom-up segmentation (greedy), and bottom-up segmentation (global). The Gecko and Bottom-up segmentation (BUS) algorithms were explained in Section 2. BUS-greedy's *y*-axis in the evaluation graph is the increase in error of the 2 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. All three algorithms used in this evaluation make use of an initial top-down pass to create the initial fine-grain segments. The minimum size of each initial cluster generated in the top down pass was 10.

4.2.2 Results and Analysis

A summary of the results of the L Method's ability to automatically determine the number of segments to return from segmentation algorithms is contained in Table 2. For each algorithm and data set, the 'reasonable' range of correct answers is listed as well as the number of segments returned by the L Method. The first three data sets are synthetic and have a single correct answer, but the other data sets have a range of 'reasonable' answers. Numbers in brackets after the reasonable range are the best answers within that reasonable range. The signature in data set #5 was fractal in nature and had no 'correct' clustering. Note that BUS-greedy and BUS-global perform identically and therefore have identical 'reasonable' answers. However, their evaluation graphs differ and the L Method returns different answers for the two algorithms.

Table 2. Results of using the L Method with three hierarchical segmentation algorithms. Numbers in brackets are the 'best' answers within a range of correct answers.

	Gecko		**Bottom-up-greedy**		**Bottom-up-global**	
Data Set	*'Correct' # of segments*	*Num of segments given*	*'Correct' # of segments*	*Num of segments given*	*'Correct' # of segments*	*Num of segments given*
1	20	20	20	20	20	25
2	20	20	20	20	20	37
3	20	N/A	20	20	20	37
4	42-~123 [42, ~56]	92	42-~123+ [42]	46	42-123+ [42]	133
5	?	32	?	14	?	13
6	44-57 [44]	45	45-53 [45]	48	45-53 [45]	31
7	9-20 [10, 17]	17	14-21 [15]	9	14-21 [15]	19

The L Method works 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. For data sets #4 and #6 it also identified a number of clusters that is very close to the 'best' value in the range of reasonable values. Gecko, which also uses a greedy evaluation metric, had the L Method suggest a number of segments within the reasonable range for all 5 applicable data sets. For the two greedy algorithms, Gecko and BUS-greedy, the L Method was able to correctly determine that the three synthetic data sets contained exactly twenty clusters. BUS-global did not fare so well. The L Method was only able to return a reasonable number of clusters for BUS-global in a single test case.

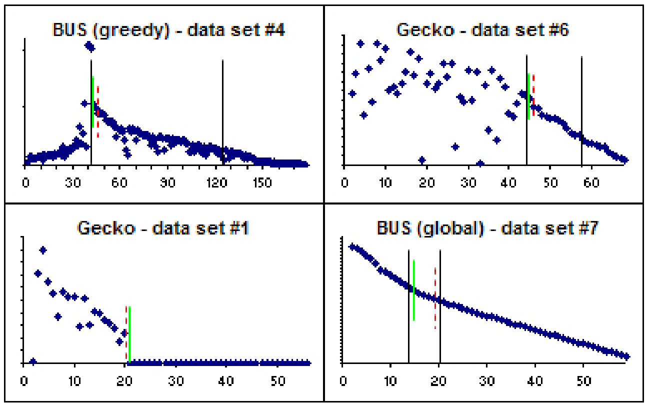

Figure 8. The reasonable range for # of clusters vs. the number returned by the L Method. (*axes*: *x*=# of clusters, *y*=evaluation metric – *lines*: *small solid line*=the 'best' # of clusters, *small dashed line*=# of clusters determined by the L Method, *large lines*=marks the boundaries of the reasonable range for the # of clusters.

Some of the evaluation graphs produced are showin in Figure 8. In the evaluation graph in the top left, the 'best' number of correct clusters was very close to being identified. Remember, that for segmentation algorithms, all data ponits to the left of the data point with the maximum value are ignored.

The evaluation graph shown in the upper-right portion of Figure 8 is also very close to identifying the 'best' number of clusters. Even though there is apparently no significant knee in this evaluation graph, the 'best' number of clusters can still be determined by the L Method. This is because the knee found by the L Method does not necessarily have to be the point of maxium curveature. It is 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…in this case it indicates that too many clusters have been merged together and the distance function is no longer as well-defined.

The poor performance of BUS-global is due to a lack of prominence in the knee of the curve compared to greedy methods (see the graph on the lower right in Figure 8). Another 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 clusters 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 clusters will most likely be returned by the L Method.

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 (most do) perform especially well. In our evaluation, the L Method was able to determine a reasonable number of clusters in 10 out of 11 instances for greedy hierarchical segmentation algorithms, and in 10 out of 12 instances for hierarchical clustering algorithms. Algorithms with global evaluation metrics are not likely to work well with the L Method because the knees in their evaluation graphs are not prominent enough to be reliably detected.

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 refinemenet algorithm explained in this paper enables the L Method to always run under optimal conditions: balanced lines on each side of the knee nomatter 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 a greater number of data sets to help further understand the algorithm's strengths and weaknesses. We will explore possible 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 directly comparing the L Method to other methods that attempt to determine the number of clusters in a data set.

6. ACKNOWLDEGEMENTS

This research is partially supported by NASA.

7. REFERENCES

[1] Baxter, R. A. & J. J. Oliver. The Kindest Cut: Minimum Message Length Segmentation. 1996.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] 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.

[8] Roth, V., T. Lange, M. Braun & J. Buhmann. A Resampling Approach to Cluster Validation. In *Intl. Conf. on Computational Statistics*, pp. 123-1298, 2002.

[9] Salvador, S., P. Chan & J. Brodie. Learning States and Rules for Time Series Anomaly Detection. Technical Report CS-2003-05, Florida Institute of Technology, Melbourne, FL, 2003. [http://cs.fit.edu/~tr/cs-2003-05.pdf].

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

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

[12] Tibshirani, R., G. Walther, D. Botstein & P. Brown. (2001) Cluster validation by prediction strength.

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

[14] 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.