Learning States and Rules for Time Series Anomaly Detection

Stan Salvador, Philip Chan, and John Brodie

Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901
{ssalvado, pkc, jbrodie}@cs.fit.edu

Abstract

The normal operation of a device can be characterized in different temporal states. To identify these states, we introduce a clustering algorithm called Gecko that can determine a reasonable number of clusters using our proposed L method. We then use the RIPPER classification algorithm to describe these states in logical rules. Finally, transitional logic between the states is added to create a finite state automaton. Our empirical results, on data obtained from the NASA shuttle program, indicate that the Gecko clustering algorithm is comparable to a human expert in identifying states and our overall system can track normal behavior and detect anomalies.

Introduction

Expert (knowledge-based) systems are often used to help humans monitor and control critical systems in real-time. For example, NASA uses expert systems to monitor various devices on the space shuttle. However, populating an expert system's knowledge base by hand is a time-consuming process. In this paper we investigate machine learning techniques for generating knowledge that can monitor the operation of devices or systems. Specifically, we study methods for generating models that can detect anomalies in time series data.

The normal operation of a device can usually be characterized in different temporal states. Segmentation or clustering techniques can help identify the various states, however, most methods directly or indirectly require a parameter to specify the number of segments/clusters in the time series data. The output of these algorithms is also not in a logical rule format, which is commonly used in expert systems for its ease of comprehension and modification. Furthermore, the relationships between these states need to be determined to allow tracking from one state to another and to detect anomalies.

Given a time series depicting a system's normal operation, we desire to learn a model that can detect anomalies and can be *easily read and modified by human users*. We investigate a few issues in this paper. First, we want a clustering algorithm that can dynamically determine a reasonable number of clusters, and hence the number of states for our purposes. These states, collected from a device, should be comparable to those identified by human experts. Second, we would like to characterize these states in logical rules so that they can be read and modified with relative ease by humans. Third, given the knowledge of the different states, we wish to describe the relationship among them for tracking normal behavior and detecting anomalies.

To identify states, we introduce Gecko, which is able to cluster time series data and determine a reasonable number of clusters (states). Gecko consists of a top-down partitioning phase to find initial sub-clusters and a bottom-up phase which merges them back together. The appropriate number of clusters is determined by what we call the L method. To characterize the states as logical rules, we use the RIPPER (Cohen 1995) classification rule learning algorithm. Since different states often overlap in the one-dimensional input space, additional attributes are derived to help characterize the states. To track normal behavior and detect anomalies, we construct a finite state automaton (FSA) with the identified states.

Our main contributions are: (1) we demonstrate a way to perform time series anomaly detection via generated states and rules that can easily be understood and modified by humans; (2) we introduce an algorithm named Gecko for clustering time series data; (3) we propose the L method for dynamically finding a reasonable number of clusters--the L method is general enough to be used with other hierarchical divisive/agglomerative clustering algorithms (Salvador and Chan 2003); (4) we integrate RIPPER and state transition logic to generate a complete anomaly detection system; (5) our empirical evaluations, with data from NASA, indicate that Gecko performs comparably with a NASA expert and the overall system can track normal behavior and detect anomalies.

Related Work

Clustering Algorithms. There are four main categories of clustering algorithms: partitioning, hierarchical, density-based, and grid-based. Partitioning algorithms, for example *K*-means, iteratively refine a set of *k* clusters. Density-based algorithms, such as DBSCAN (Ester et al. 1996), are able to efficiently produce clusters of arbitrary shape and are also able to handle noise. If the density of a region is above a specified threshold, it is assigned to a cluster, otherwise it is considered to be noise. Hierarchical algorithms can be agglomerative and/or divisive. The agglomerative (bottom-up) approach repeatedly merges two clusters, while the divisive (top-down) approach repeatedly splits a cluster into two. Our Gecko algorithm is similar to the hierarchical Chameleon (Karypis, Han, and Kumar 1999) clustering algorithm, but with constraints during the merging phase so it can be applied to time series data. Grid-based algorithms such as WaveCluster (Seikholeslami, Chatterjee, and Zhang 1998) reduce the clustering space into a grid of cells which enables efficient clustering of very large datasets.

Segmentation Algorithms. Segmentation algorithms usually take time series data as input and produce a Piecewise Linear Representation (PLR) as output. PLR is a set of consecutive line segments that tightly fit the original data points. Segmentation algorithms are somewhat related to clustering algorithms in that each segment can be thought of as a cluster. However, due to the linear representation bias, segmentation algorithms are much more effective at producing fine grain partitioning, rather than a smaller set of segments that represent natural clusters.

There are three common approaches (Keogh et al. 2001). 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 is reached, or an error threshold is reached. Third, the Bottom-up approach starts off with *n*/2 segments, the 2 most similar adjacent segments are repeatedly joined until either the desired number of segments, or an error threshold is reached.

Determining the Number of Segments/Clusters. 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 (Smyth 1996) 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. Permutation tests (Vasko & Toivonen 2002) are able to prevent segmentation algorithms from creating a PLR that over-fits the data. Resampling (Monti et al. 2003) 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."

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 (Tibshirani, Walther, and Hastie 2000) and prediction strength (Tibshirani et al. 2001).

Anomaly Detection. Much of the work in time series anomaly detection relies on models that are not easily readable and hence cannot be modified by a human for tuning purposes. Examples include a set of normal sequences (Dasgupta and Forrest 1996) and adaptive resonance theory (Caudell and Newman 1993).

Approach

The input to our overall anomaly detection system is "normal" time series data (like the graph at the top left corner of Figure 1). The output of the overall system is a set of rules that implement state transition logic on an expert system, and are able to determine if other time series signatures deviate significantly from the learned signature. Any deviation from the learned "normal" model is considered to be an anomaly. The overall architecture of the anomaly detection system, depicted in Figure 1, consists of three components: clustering, rule generation (characterization), and state transition logic. The clustering phase is performed by our newly-developed clustering algorithm "Gecko," which is designed to identify distinct phases in a time series. Then rules are created for each state by the RIPPER algorithm (Cohen 1995). Finally, rules are added for the transitions between states to create a finite state automaton. The three steps in our approach are detailed in the next three subsections.

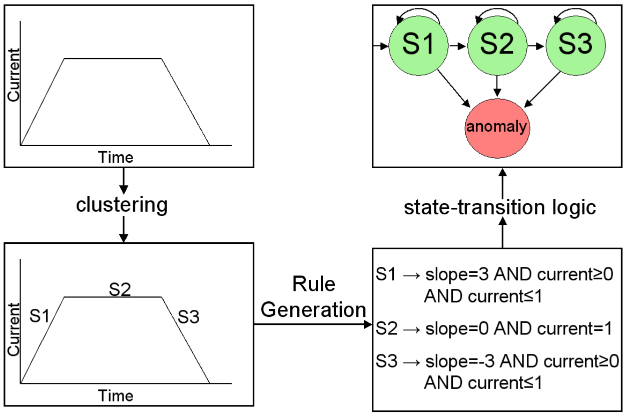
Current
Time
clustering
S1
S2
S3
Current
Time
Rule Generation
S1 → slope=3 AND current≥0 AND current≤1
S2 → slope=0 AND current=1
S3 → slope=-3 AND current≥0 AND current≤1
state-transition logic
S1
S2
S3
anomaly

Figure 1. Main steps in time series anomaly detection.

Gecko – Data Clustering

While segmentation algorithms typically create only a fine linear approximation of time series data, Gecko divides a time series into clusters. This number of clusters is determined by the algorithm and requires no user input. The Gecko algorithm consists of three phases, as depicted in Figure 2. The first phase creates many small sub-clusters. The second phase repeatedly merges the two most similar clusters. Phase 3 determines the number of clusters to return.

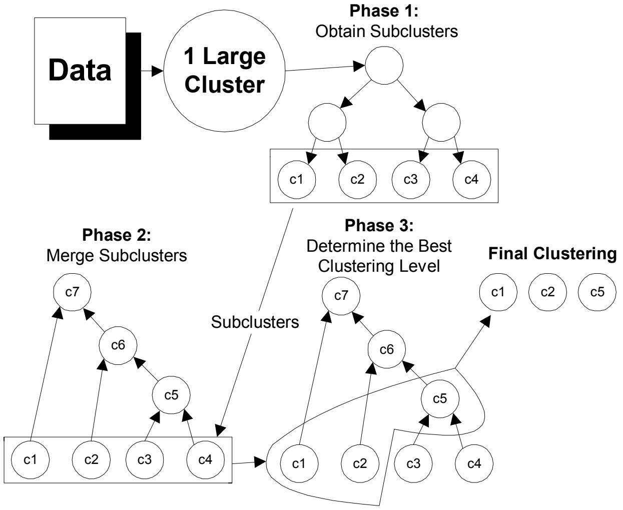

Figure 2. Overview of the Gecko Algorithm.

Phase 1: Create Sub-Clusters. In the first phase, many small sub-clusters are created by a method that is similar to the one used by Chameleon (Karypis, Han, and Kumar 1999), except that Gecko forces cluster boundaries to be non-overlapping in the time dimension. The sub-clusters are created by initially placing all of the data points in a cluster, and repeatedly splitting the largest cluster until all of the clusters are too small to be split again without violating the minimum possible cluster size *s*.

```
The Gecko Algorithm (overview)
Input:  D          // time series data
        s          // the minimum cluster size
Output: c* clusters

Phase 1:
1. build a k-nearest neighbor graph of D (k=2*s)
2. recursively bisect the graph until no bisections can be
   made without creating a cluster smaller than s

Phase 2:
3. recursively merge the sub-clusters together until only one
   cluster remains - a dendrogram is created

Phase 3:
4. find c*, an appropriate number of clusters to return, by
   using the L method.
5. extract c* clusters from the dendrogram and return them
```

To determine how to split the largest cluster, a *k*-nearest neighbor graph is built in which each node in the graph is a time series data point, and each edge is the similarity between two data points. Only the slopes of the original values are used to determine similarity, and not the original values themselves. Using only the slope will tend to produce sub-clusters that are straight lines. For more specific details about phase 1, please refer to (Salvador, Chan, and Brodie 2003) due to space constraints.

Phase 2: Repeatedly Merge Clusters. In phase 2, the most similar pair of adjacent (in time) clusters are repeatedly merged until only one cluster remains. To determine which adjacent pair of clusters are the most similar, representative points are generated for each cluster and the two adjacent clusters with the closest representative points are merged. A single representative point is able to accurately represent every point in a cluster because each cluster is internally homogeneous. The representative point of a cluster contains a value for the slope of every original attribute in the data other than time. Clustering by the slope values causes the time series to be divided into flat regions. Segmentation also relies exclusively on slope: if a minimum-error line (segment) is well fitted to a set of points it means that the segment has a consistent slope.

If raw slope values are used in the representative points, then the “distance” between clusters with slope values 100 and 101 would be the same as the distance between clusters with slope values 0 and 1. Differences in slopes that are near zero need to be emphasized because the same absolute change in slope can triple a small value, and be an insignificant increase for a large value. Relative differences between slopes cannot be measured by the percentage increase because in the preceding example, the percentage increase from 0 to 1 is undefined. Gecko uses representative values of slopes to determine the “distance” between two slopes by using the equation:

$$\text{Representative Slope} = \begin{cases} \ln(slope+1) & if\ slope \geq 0 \\ -\ln(-slope+1) & if\ slope < 0 \end{cases}$$

This equation emphasizes slopes near zero and decreases the effect of changes in slope when the slope values are large. Whenever a slope value is squared, its representative slope value (approximately) doubles. In the preceding example of comparing 2 pairs of clusters with slopes {100, 101} and {0, 1} the representative values of their slopes are {4.615, 4.625} and {0, 0.693}. This accurately reflects the relative difference between raw slopes and not the absolute difference.

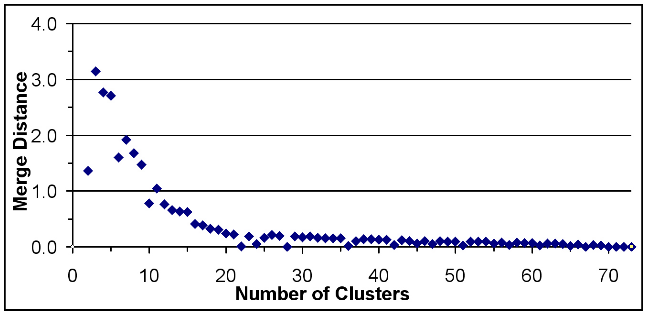

Figure 3. A sample # of clusters vs. merge distance graph.

Phase 3: Determine the Best Clustering Level. To determine a good number of clusters to return, the distances of all merges (all the way to a single cluster) during phase 2 are analyzed. The basic shape of the ‘*# of clusters* vs. *merge distance*’ graph is shown in Figure 3. In this graph, the *x*-axis is the number of clusters from ‘2’ to ‘the number of sub-clusters generated by phase 1’. The *y*-axis is the distance of the two closest clusters when there are *x* clusters. Each data-point is the distance of a single merge, and the entire graph is generated in only once pass of the clustering algorithm. The majority of ‘*# of clusters* vs. *merge distance*’ graphs have three distinctive areas: a rather flat region to the right, a near-vertical region to the left, and a curved transition area in the middle.

Starting from the right end, where the phase 2 merging process begins, there are many very similar clusters that should be merged. Another distinctive area of the graph is on the far left side where the merge distances grow very rapidly. 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. In this region, too many merges have already been performed and the optimal clustering has been passed. The optimal number of clusters is therefore in the curved area, or the “knee” of the graph. This 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.

The regions to both the right and the left of the curved section of the graph (see Figure 3) are approximately linear. If a line is fitted to the right side and another line is fitted to the left side, then the area between those two lines will be in the transition area and can be used as the number of clusters to return. Figure 4 depicts an example. To find these two lines, we choose the pair of lines that most closely fit the curve. 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 short, the other is long 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.

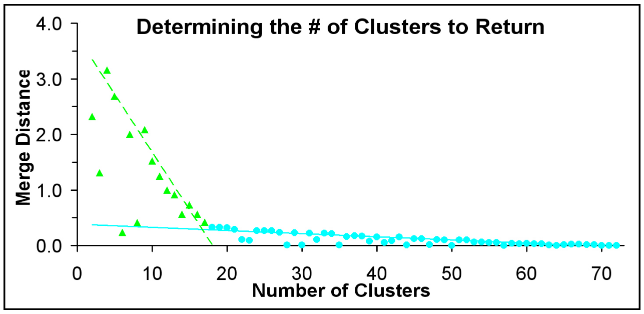

Figure 4. Finding the number of clusters by the L method.

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:

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

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:

$$c^* = \arg\min_c \quad RMSE_c \qquad [2]$$

The location of the knee at x=c* is used as the number of clusters to return. This method to determine the number of clusters to return is general, and can also be used to determine the number of clusters in other hierarchical clustering and hierarchical segmentation algorithms, as shown in (Salvador and Chan 2003).

RIPPER – Rule Generation

We have adapted RIPPER (Cohen 1995) to generate human readable rules that characterize the states identified by the Gecko algorithm. The RIPPER algorithm is based on the Incremental Reduce Error Pruning (IREP) (Furnkranz and Wildmer 1994) over-fit-and-prune strategy. The IREP algorithm is a 2-class approach, where the data set must first be divided into two subsets. The first subset contains examples of the class whose characteristics are desired (the positive example set) and the other subset contains all other data samples (the negative example set). Our implementation of RIPPER acts as an outer loop for the IREP rule construction.

The input to RIPPER is the data produced by Gecko which contains time series data classified into c* states. RIPPER will execute the IREP algorithm c* times, once for each state. At each execution of IREP, a different state is considered to be the positive example set and the rest of the states form the negative example set. This creates a set of rules for each state. To describe the relationship among these states, state transition logic is identified as discussed in the following section.

State Transition Logic

The upper right-hand quadrant of Figure 1 depicts a simplified state transition diagram for a signal containing just three states. The state transition logic is described by three rules for each state corresponding to each of the three possible state transition conditions on each input data point:

- IF input matches current state's characteristics THEN remain in current state.
- IF input matches the next state's characteristics THEN transition to the next state.
- IF input matches neither the current state's nor the next state's characteristics THEN transition to an anomaly state.

The antecedent condition for each state is obtained from the RIPPER rule generation process. The state transition logic simply needs to glue together the proper antecedents to formulate the above three transition rules for each state.

Before an anomaly state is entered, one of two additional criteria must be satisfied: either (1) the number of consecutively observed anomalous values must exceed a specified threshold; or (2) the total number of anomalous values observed has exceeded another threshold. Thus, an anomalous condition is not annunciated unless the observed values have been improper for some length of time. Similar logic is provided for the transition from a normal state to its normal successor to prevent premature state transitions.

Empirical Evaluation

The goal of this evaluation is to demonstrate the ability of the Gecko algorithm to identify states in real time series data, and also to show that our overall system is able to detect anomalies. The data used to evaluate Gecko and the overall anomaly detection system is 10 time series data sets obtained from NASA. The data sets are signatures of a valve from the space shuttle.

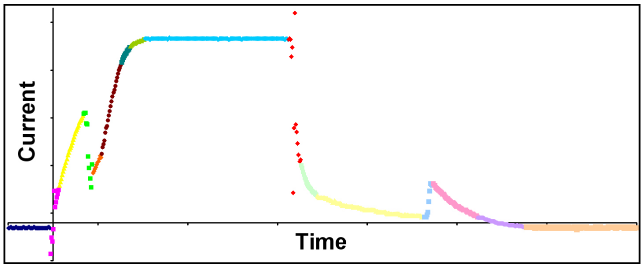

Figure 5. A data set after being clustered by Gecko.

Each data set contains between 1,000 and 20,000 equally spaced measurements of current. These 10 data sets contain signatures of valves that are operating normally, and also signatures of valves that have been damaged. The current method used to test these valves requires a human expert to compare a valve's signature to a known normal signature, and determine if there is any significant variation. We would like to demonstrate that Gecko is able to cluster time series signatures into important phases, and that our anomaly detection system is able to determine if a valve is operating normally.

Identifying States with Gecko

Procedures and Criteria. First, Gecko and a valve expert from NASA independently cluster the 10 data sets. The expert is given an unclustered graph of each data set. Based on his or her knowledge of the valve's states, the human expert is asked to draw lines between cluster boundaries. This allows us to determine if the number of clusters determined by the Gecko algorithm is comparable to the number of clusters produced by the human expert. Second, both Gecko and an existing algorithm cluster the 10 data sets. Without knowing which output is from which algorithm, a NASA engineer will then rate the quality of each clustering from 1 to 10. The existing algorithm that is used is bottom-up segmentation (BUS). The number of clusters returned by BUS is set to be the same number that Gecko returns. Finally, the valve expert is asked to go over all of the Gecko data sets that he or she rated in the second step, and explain the evaluation that was given. Gecko was run with the default parameter for each data set: minimum cluster size s=10.

Table 1. Clusters produced by Gecko and a human expert.

	Gecko	NASA Human Expert	
Data Set	*# of clusters*	*# of clusters*	*Reasonable Range*
1	16	11	9-20
2	16	10	9-20
3	14	10	9-20
4	12	10	9-20
5	13	7	(6-15)
6	10	5	(5-10)
7	7	6	(6-11)
8	16	10	(9-19)
9	16	12	(10-20)
10	15	11	(9-16)

Results. The first part of Gecko's evaluation was to compare the number of clusters it produced to the number produced by an expert human. A summary of the results is shown in Table 1.

Gecko was able to identify a number of clusters that was within the range specified by the expert to be a 'reasonable range' (for datasets 5-10 the expert did not provide a range and we extrapolated from his hand-clustering and his ranges for data sets 1-4). The human expert consistently created clusterings with fewer clusters than the Gecko algorithm. However, the clusterings are actually quite similar. Gecko identifies the same major clusters as the valve expert, but also produces several 'transition' clusters between them. A more detailed evaluation of the L Method's ability to determine the number of clusters for more diverse data sets can be found in (Salvador and Chan 2003).

The next task performed by the NASA engineer was to rate the clusterings produced by Gecko and BUS. Table 2 contains the clustering quality scores for Gecko and BUS. Gecko's average score was 9.5, while the bottom-up segmentation algorithm's average score was only 4.3. Notice that Gecko often receives a perfect clustering score (which signifies a clustering as good as the human expert's clustering) even though it returns fewer clusters than the human expert. For example, Gecko produced nearly twice as many clusters as the human expert for data set 5, and Gecko still got a perfect rating. This suggests that there is often a range of "very good" numbers of clusters to return, rather than a single correct number.

Table 2. Clustering quality of Gecko and BUS

Data Set	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	Avg
Gecko	10	10	9	10	10	10	8	9	9	10	**9.5**
BUS	2	3	3	3	3	3	8	5	7	6	**4.3**

The final part of Gecko's evaluation was a discussion with the NASA engineer about why he gave each score. According to the engineer, BUS divides regions of high slope into too many clusters. BUS merges clusters together by keeping the root-mean squared error of the best fit lines to a minimum. This method measures error vertically, and as a consequence, lines that are nearly vertical may seem visually to be a nearly perfect fit, but the vertical distances from the points to the line can be huge.

Overall System (FSA)

Procedures and Criteria. In order to test whether the anomaly detection system works correctly we performed three kinds of tests: (1) Self-tracking: Use 90% of the data points to create rules, and then use 100% of the data fed into the expert system to see if the state transitions occur correctly, without detecting any anomalies. (2) Normal operation: Use all of a normal valve's data to learn its signature, and then monitor another valve that is also operating normally. This case should also not trigger any anomalies. (3) Detecting anomalies: Use all of a properly functioning valve's data to learn its normal signature, and then take signatures of valves that are damaged slightly and run them through the anomaly detection system. The damaged valves should trigger anomalies.

Self-tracking Results. The baseline test of the anomaly detection system is to train the model with 90% of the data, and seeing if 100% of the data can be tracked without triggering an anomaly. The results of this test are shown in Table 3. An error point in Table 3 is any point that is unexpected in the state transition logic. This means that the point is neither in the current state or the following state. Time series data often contains noise and minor variations. For this reason, anomalies must not be triggered by only a single data point that does not agree with the model contained in the FSA. By using a threshold counter, an anomaly will only be reported after a certain number of consecutive error points. The last row in Table 3 shows what the minimum consecutive error threshold (*CE*) must be set to for the anomaly detection system to not report an anomaly. A value of 1 in this last column means that the anomaly detection system will correctly not report an anomaly as long as *CE* ≥ 1.

Table 3. Self-tracking of a time series.

Data Set	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	Avg
Error Pts (%)	1.1	0.8	0.7	0.5	0.0	0.4	0.3	0.2	0.4	1.1	**0.6**
Min. Error Threshold	2	2	1	1	0	1	1	1	1	21	**4.0**

In this experiment, both the "consecutive transition" (*CT*) and the "consecutive error" (*CE*) thresholds were set to zero. This causes every possible state transition to be made and every error point triggers an anomaly. This enabled easy computation of the number of error points. Data set number 10 performs poorly in this test because the FSA transitions prematurely near the end of its signature and starts reporting many anomalies, the results for this data set can be improved by increasing *CT* to prevent it from transitioning too early on a single spurious data point.

Normal Operation Results. This test is to show that the anomaly detection system's model of the normal signature is general enough to recognize that an untrained normal time series contains no anomalies. In this test, the anomaly detection system trained on data set 1, and then tested on data set 2. Both of these data sets are of normally operating valves that contain minor (but visible) differences. The "consecutive transition" threshold (*CT*) parameter was set to 2, and *CE* was set to 10 (minimum possible cluster size *s*=1*0*). This means that two consecutive points believed to be in the next state are needed to perform a state transition and ten consecutive points believed to be errors are needed to declare that the time series contains anomalies.

The system was able to successfully transition through the states, without detecting any anomalies. Of 979 data points, 61 (2.6%) were error points--they were not believed to belong to the current state, nor to be transition points belonging to the following state. However, since a consecutive number of errors greater than *CE* was never encountered, an anomaly was never triggered.

Detecting Anomalies Results. This final test is to show that our system is capable of detecting when a time series differs significantly from the learned model. In this test, two data sets containing time series signatures of valves operating normally (data sets 1 and 2) were used to develop the normal models. Each normal model was then run against the remaining anomalous data sets (data sets 3…10).

For each of the 16 tests, the anomaly detection system correctly determined that the signatures contained anomalies. Additionally, the system was able to inform the user of the state number where the signature differs from the model. Thus, the system does not only give a yes/no answer to whether a time series contains anomalies, but it is also able to explain to the user where the anomaly occurred. Also, because the rules generated by RIPPER are in a human-readable format, the user can look at the rule for the state where the error occurred and understand exactly why the system reported the anomaly.

Concluding Remarks

We have detailed our approach to time series anomaly detection by discovering and characterizing the states of a time series, and performing transition logic between these states to construct a finite state automaton, run on an expert system, that can be used to track normal behavior and detect anomalies. The proposed Gecko clustering algorithm is designed to cluster time series data, and uses our proposed L method to determine a reasonable number of clusters efficiently. The rules generated for each state by the RIPPER algorithm can be *easily understood and modified by humans*. (Moreover, the generated rules can be in a format used by the SCL expert system shell at ICS, which is our collaborator on this NASA project)

Our empirical evaluations have shown that the L method used by the Gecko algorithm returns a number of clusters that is similar to the number that is generated by a human expert. When the human expert was asked to rate Gecko's clusterings from 1-10, Gecko's clusterings were given perfect ratings on 6 of 10 data sets. A perfect rating signifies that Gecko's clustering is equally as good as the human expert's clustering. For comparison, the bottom-up segmentation algorithm was also tested, and was only given an average rating of 4.3. The overall anomaly detection system was able to detect anomalies in every signature that was from a 'damaged' valve, and was also able to monitor a 2nd normal valve without detecting any anomalies.

We plan to further evaluate our approach with more datasets from NASA; issues include building a model from multiple datasets collected at different times and datasets with different measurements. We plan to continue studying how the L method performs with other hierarchical clustering algorithms (Salvador and Chan 2003). To dynamically set the thresholds used in the state transition logic, we can investigate holding out part of the training data and find thresholds that prevent errors on the unseen portion of the data.

Acknowledgements

This research is partially supported by NASA. We thank Bobby Ferrell and Steven Santuro at NASA for providing the data sets, helpful comments, and clustering evaluations. We also thank Brian Buckley and Steve Creighton at ICS for help integrating our algorithms into their SCL expert system.

References

Caudell, T. and Newman, D. 1993. An Adaptive Resonance Architecture to Define Normality and Detect Novelties in Time Series and Databases. In *Proc. IEEE World Congress on Neural Networks*, 166-176. Portland, OR.

Cohen, W. 1995. Fast Effective Rule Induction, In *Proc. of the 12th Intl. Conf. on Machine Learning*, 115-123. Tahoe City, CA.

Dasgupta, D. and Forrest, S. 1996. Novelty Detection in Time Series Data using Ideas from Immunology. In *Proc. Fifth Intl. Conf. on Intelligent Systems*, 82-87. Reno, NV.

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. 2nd Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, 226-231. Portland, OR.

Furnkranz, J. and Wildmer, G. 1994. Incremental Reduced Error Pruning. In *Proc. of the 11th Intl. Conf. on Machine Learning*, 70-77, New Brunswick, NJ.

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

Keogh, E.; Chu, S.; Hart, D.; and Pazanni, M. 2001. An Online Algorithm for Segmenting Time Series. In *Proc. IEEE Intl. Conf. on Data Mining*, 289-296. San Jose, CA.

Monti, S. et al. 2003. Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data. *Machine Learning*, 52(1-2):91-118.

Salvador, S. and Chan, P. 2003. Determining the Number of Clusters/Segments in Hierarchical Clustering/Segmentation Algorithms, Technical Report, CS-2003-18, Dept. of Computer Sciences, Florida Institute of Technology.

Salvador, S.; Chan, P.; and Brodie, J. 2003 Learning States and Rules for Time Series Anomaly Detection, Technical Report, CS-2003-05, Dept. of Computer Sciences, Florida Institute of Technology.

Seikholeslami, G.; Chatterjee, S.; and Zhang, A. 1998. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. *Proc. of the 24th Intl. Conf. on Very Large Databases (VLDB)*, 428-439. New York City, NY.

Smyth, P. 1996. Clustering Using Monte-Carlo Cross-Validation. In *Proc. 2nd Knowledge Discovery and Data Mining (KDD)*, 126-133. Portland, OR.

Tibshirani, R. et al. 2001. Cluster Validation by Prediction Strength, Technical Report, 2001-21, Dept. of Biostatistics, Stanford Univ.

Tibshirani, R.; Walther, G.; and Hastie, T. 2000. Estimating the number of clusters in a dataset via the Gap statistic, Technical Report, 208, Dept. of Biostatistics, Stanford Univ.

Vasko, K. and T. Toivonen. 2002. Estimating the number of segments in time series data using permutation tests. In Proc. *IEEE Intl. Conf. on Data Mining*, 466-473. Maebashi City, Japan.