Trajectory Boundary Modeling of Time Series for Anomaly Detection

Matthew V. Mahoney and Philip K. Chan

Computer Science Dept.
Florida Institute of Technology
Melbourne FL 32901

{mmahoney,pkc}@cs.fit.edu

ABSTRACT

We address the problem of online detection of unanticipated modes of mechanical failure given a small set of time series under normal conditions, with the requirement that the anomaly detection model be manually verifiable and modifiable. We specify a set of time series features, which are linear combinations of the current and past values, and model the allowed feature values by a sequence of minimal bounding boxes containing all of the training trajectories. The model can be constructed in $O(n \log n)$ time. If there are at most three features, the model can be displayed graphically for verification, otherwise a table is used. Test time is $O(n)$ with a guaranteed upper bound on computation time for each test point. The model compares favorably with anomaly detection algorithms based on Euclidean distance and dynamic time warping on the Space Shuttle Marrotta fuel control valve data set.

Keywords

Time series anomaly detection, Machine health monitoring, Path model, Box model, Rule Learning, NASA.

1. INTRODUCTION

In 1996 an Ariane 5 rocket self destructed during launch because the primary and backup flight control units had identical software errors. In each processor, a 64 bit floating point number was assigned to a 16 bit integer, raising an unhandled Ada overflow exception and halting it [1]. In 1999 the Mars Climate Orbiter was lost when engineers sent navigation commands using English units, while the spacecraft was expecting metric units [2]. In 2004, half of the data sent by the Huygens probe to Titan was lost because one of two receiver channels on the Cassini mother craft orbiting Saturn was not turned on due to a software error [14].

We are given the task of automating the detection of mechanical failures in the Marrotta fuel control valves used in the space shuttle. Because not all failure modes can be anticipated, this is an ideal task for time series anomaly detection: train a model on known good data, estimate the probability distribution, and assign a likelihood-based score to new sensor data. However, NASA is keenly aware of the consequences of software errors on a manned spacecraft. Therefore a requirement of our project is that the model be transparent. It is not enough that we demonstrate the ability to detect anomalies caused by simulated failures in the lab. Engineers also want to know *what* the modeler learned, and if necessary, manually update the model using domain specific knowledge. Unfortunately, many good time series anomaly detection algorithms produce opaque models that are difficult to analyze.

Our goal is to produce an anomaly detection system whose model is transparent. In addition, testing must be online, fast, and generalize when given more than one training series. By online, we mean that each test point receives an anomaly score, with an upper bound on computation time. We accept that there is no "best" anomaly detection algorithm for all data, and that many algorithms have *ad-hoc* parameters which are tuned to specific data sets. Therefore our subgoal is to provide tools to make this tuning easier on a given data set. The software that allows this capability is not directly discussed in this paper.

Our approach is to offer a set of models based on feature trajectory paths, because these models can be visualized in two or three dimensions, or coded as rules which can be edited in higher dimensions. A *feature* is defined as a linear combination of present and past values (a digital filter), for example, a time lagged copy, a derivative, or a smoothed signal. Thus, a feature is also a time series. Given d features, a signal traces a path or trajectory through d-dimensional feature space. The idea is that a test series should follow a similar trajectory to that of a known good training signal, or at least be near the training trajectory at all times. An engineer may choose to approximate the trajectory using straight line segments or a sequence of boxes for performance reasons. There may also be more than one training series, in which case we can construct a model which encloses all of the trajectories.

Our main contributions include:

- we propose two anomaly detection methods based on models that are transparent/editable, generalizable from multiple training time series, efficient during testing, and provide online scoring during testing;
- our empirical results from the NASA shuttle valve data indicate that our methods can detect similar or more abnormal time series than three existing methods.

KDD-05, Aug. 21, 2005, Chicago IL, USA..

The rest of the paper is organized as follows. In Section 2 we discuss related work. In Sections 3 and 4 we introduce path and box modeling respectively, along with efficient algorithms for generating approximations. In Section 5 we present experimental results with the NASA valve data set. In Section 6, we conclude.

2. RELATED WORK

One view of time series anomaly detection is that of a machine learning or modeling task. Given a training set X of time series with an unknown probability distribution P, the task is to estimate P. Then given a new time series y, we assign an anomaly score inversely related to $P(y)$. Ypma [15] surveys some important techniques, such as Bayesian models, neural networks, and support vector machines, and applications to the detection of failures in rotating machinery using vibration sensors

Dasgupta and Forrest [4] uses an immunological approach. A time series is quantized and chopped into fixed length strings of several symbols. A random set of strings is generated. Any strings which match the training data are removed. The remaining strings form an anomaly model. If a test signal matches any strings in the model, then an alarm is signaled. This technique was shown to detect simulated failures in a milling machine.

Keogh approaches the problem as that of finding a dissimilarity function $D(x, y)$ between a (normal or good) training series x and a test series y [7]. Viewed this way, we avail ourselves of the vast body of research in related data mining topics such as classification, clustering, and search. The simplest measure is Euclidean distance:

$$D_{EUCLID}(x, y)^2 = \sum_{i=1}^{N} (x_i - y_i)^2 \qquad (1)$$

where both series have length N and $x_1, x_2, ..., x_N$ are the N values of x. In some applications, we normalize x and y to have zero mean and unit standard deviation. Two disadvantages of this measure are that the series must have equal length and it is sensitive to shifts in time. Dynamic time warping (DTW) overcomes these problems by finding the minimum Euclidean distance when the data points of both series may be shifted arbitrarily in time (but maintained in order). DTW is defined recursively as follows:

$$DTW(x, y) = \sqrt{D(x_1^m, y_1^n)} \qquad (2)$$

where

$$D(x_1^i, y_1^j) = (x_i - y_j)^2 + \min[D(x_1^{i-1}, y_1^j), D(x_1^i, y_1^{j-1}), D(x_1^{i-1}, y_1^{j-1})]$$

and x_1^i means the sequence $x_1, x_2, ..., x_i$ and $D(x, y)$ is infinite if either x or y is empty. A *warp path* is the set of (i,j) from (1,1) to (m,n) such that if all x_i are aligned with y_j by shifting them in time, then $DTW(x, y) = D_{EUCLID}(x, y)$.

A disadvantage of DTW is that computation time is $O(mn)$. Various fast approximations have been proposed. For example, Salvador [10] describes FastDTW, an approximation to DTW in which the warp path is estimated as successively higher resolutions and the search is constrained within a radius of the previous estimate.

Many other distance measures have been proposed. In an exhaustive test, Keogh and others at UCR implemented about 50 proposed distance measures published over a 10 year period and evaluated them on a variety of data mining tasks on a large corpus of time series from diverse domains [7]. The rather surprising finding is that while many of the proposed measures improve over existing techniques on the specific data sets on which they were tested, none did better than normalized Euclidean distance over the entire data set.

Keogh also proposes a very general method which does outperform Euclidean distance on this diverse set: a compression dissimilarity measure, or CDM [8], defined as:

$$CDM(x, y) = \frac{C(xy)}{C(x) + C(y)}$$

where $C(x)$ is the compressed size of a symbolic (SAX) representation of x, saved as a file and compressed with an off-the-shelf compressor such as *gzip*. The idea is that CDM estimates the information shared by x and y. If the two series are identical, then a compressor can store y as a reference to x, so $C(xy) \approx C(x)$ and $CDM(x, y) \approx 0.5$. If x and y are unrelated, then the compressor cannot use knowledge from x to model y, so $C(xy) \approx C(x) + C(y)$ and $CDM(x, y) \approx 1$.

2.1 Feature Trajectory Models

Some proven and broadly applicable techniques such as CDM and neural networks suffer from opacity. It is not at all clear from the state of a data compression program or the trained weights of a neural network exactly what has been learned. Our work is based on trajectory modeling in feature space as described by Povinelli et. al. [9]. Povinelli extracted d features of a time series, which are simply time-lagged copies of the data delayed by t, $2t$, $3t$, ... dt, and d and t are parameters. The density in d-dimensional feature space is modeled by clustering the training points and using a Gaussian mixture model to approximate the clusters. A test point is evaluated by its distance (in standard deviations) from the nearest cluster. The model was shown to classify phonemes in speech, detect arrhythmias in ECG traces, and detect mechanical failures in a motor simulation.

Generating a Gaussian mixture model requires a slow, iterative process. Vlachos et. al. [13] describe a minimum bounding rectangle (MBR) clustering algorithm that runs in $O(n \log n)$ time that is nearly identical to the one used in our system. A sequence of n points in feature space is first approximated by a sequence of $n - 1$ boxes, each enclosing a pair of adjacent points. Then pairs of adjacent boxes are merged by greedily selecting the pair that minimizes the increase in volume after merging. The algorithm for modeling the sequence of n points $x_1, x_2, ..., x_n$ using k boxes is as follows:

```
MBR(x1...n, k)
  For each i in [1, n-1] do
    xi := merge(xi, xi+1)
  Delete xn
  While n > k do
    Find i minimizing ΔV =
       V(i, i+1) – V(i) – V(i+1)
       (minimize increase in volume)
    xi := merge(xi, xi+1)
    Delete xi+1
  Return x = x1...k
```

Fig. 1. MBR Algorithm.

In the MBR algorithm, `merge(x, y)` means to replace points or boxes *x* and *y* with the smallest box that encloses both, `V(i)` means the volume of x_i, and `V(i, i+1)` means the volume of `merge(`x_i`, `x_{i+1}`)`. ΔV is the increase in volume that would result from merging. Deleting an element x_i implicitly decrements *n*.

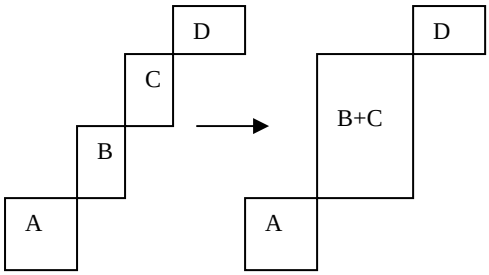

Fig. 2. Merging boxes B and C in the MBR algorithm.

MBR can run in $O(n \log n)$ time by storing the boxes in a heap ordered by ΔV, the increase in volume that would result from merging it with the next box. In a heap, the elements are stored in a balanced binary tree such that at each node the parent is smaller than the two children. Each node x_i also stores pointers to x_{i-1} and x_{i+1} to form a doubly linked list. When the box at the root of the heap is merged with its neighbor, the two old boxes are removed from the heap, the merged box is inserted, and ΔV of the two neighbors of the new box are updated, requiring them to be sifted up or down the heap. Each of the heap operations takes $O(\log n)$ time.

2.2 Gecko

In our earlier work on the NASA valve data [5], we used the Gecko algorithm [11] to create a bounded rectangle model. The Gecko model is more complex and less efficient than MBR in the training phase, but our interest is in the correctness of the model and efficiency in the testing phase. Gecko uses 3 dimensions of feature space: the original signal and the first and second derivatives, each of which is smoothed by a low pass filter. The trajectory is then segmented in feature space using a bottom-up clustering algorithm. Next, RIPPER [3] is used to generate a minimal rule set which separates the clusters. Each rule corresponds to one surface of one box, for example "*if segment = 3 then feature2 < 2.5*". It is possible to define one segment by several boxes, and some boxes may be open on some sides. Gecko, like MBR, satisfies our criteria that the model be comprehensible. The feature space can either be visualized in three dimensions, or expressed as a set of *if...then* rules.

During testing, a state machine is constructed such that each state corresponds to one trajectory segment, plus one error state. A transition to the next state occurs if the number of consecutive points satisfying the rules for the new state (falling within one of the bounding boxes) exceeds a threshold. An error occurs if the number of consecutive points satisfying neither the current nor next state exceeds a second threshold. Both thresholds are user defined parameters.

Gecko has been extended to handle multiple training series. First, the series are aligned by DTW or FastDTW. Next, the aligned series are averaged. Then the averaged series is segmented as before. Finally RIPPER is applied to separate the points in the original series that align with different segments in the merged series.

3. PATH MODELING

Our work in time series modeling falls between two extremes. At one end, we have a single training series, and we compute the distance from it using some function. At the other extreme, we have a large set of training sequences (or a single series with thousands of cycles) which we model using a probability distribution in a feature space and then estimate the probability of the test series. The NASA valve data set is one example of a data set that falls in the middle. We have one to four "normal" training series from which we generalize to a model. Our approach is to construct a model that encloses all of the training trajectories and the space "between" them.

We describe two representations that approximate this space, *path modeling* and *box modeling*. For path modeling, we store the training trajectories and test whether the sensor data falls between or near these paths. For box modeling, we construct a sequence of boxes enclosing all of the training paths, and test whether a test point falls within or near these boxes. We describe path modeling in this section, and box modeling in Section 4.

For the case of a single training path *x* in *d*-dimensional feature space, and a point y_i in a test series *y*, we could assign an anomaly score $D(x, y_j)$ equal to the square of the Euclidean distance between y_j and the nearest point in *x*.

$$D(x, y_j) = \min_{i \in [1,n]} \sum_{k=1}^{d} (x_{ik} - y_{jk})^2 \qquad (3)$$

where x_{ik} denotes the value of the *k*'th feature of the *i*'th point in *x*. This measure would have two problems. First it is inefficient because the testing time would be $O(dn)$ per test point (or $O(n)$ best case if we test the nearest points first). Second, the score would be nonzero even for the case of a test path following the training path exactly, because *x* is sampled and y_i could fall between the sample points in *x*. Addressing the latter problem by increasing the number of samples would make the first problem worse.

Our approach is to model *x* using a piecewise linear approximation of $k - 1$ straight line segments defined by *k*

vertices, where k is a parameter. Then we define $D(x, y_j)$ to be the square of the Euclidean distance between y_j and the nearest point in the approximation of x. The computation time is now $O(kd)$, where $k << n$. Computing the distance between a point and a line segment is more complex than computing the distance between two points, but is still $O(d)$.

Depending on the domain, we might require that the test signal follow the same trajectory as the training data in the same order. This restriction, which we call *sequential testing*, is used by Gecko and is appropriate when we require the training and test series to have the same overall shape, while still allowing time shifts. Suppose that the line segment (x_i, x_{i+1}) is the closest segment to test point y_j. Then it is only necessary to test the next point, y_{j+1} by computing the distance to the current and next segments, (x_i, x_{i+1}) and (x_{i+1}, x_{i+1}). We maintain i as a state variable and set it to the index of the closest segment. The time to compute $D(x, y_j)$ is now $O(d)$. Other variations are possible, such as also testing the previous segment to allow backwards movement.

Path modeling can be extended to multiple training series in a number of ways. For example, we could use 1-nearest neighbor modeling, in which the anomaly score is the square of the distance to the nearest path. If our training set is limited, it may be desirable to test whether a point lies "between" the training paths. Depending on how we define "between", this can lead to difficult calculations. We use the following definition, which is an easy to compute approximation. Given p paths and a test point y_j, we find the nearest point on each path, and then find the smallest box that will enclose all p nearest points. If y_j is inside this box, its anomaly score is zero. Otherwise its score is the square of the Euclidean distance to the box (Fig. 3). For a single path, this reduces to finding the minimum Euclidean distance from the test point to the path.

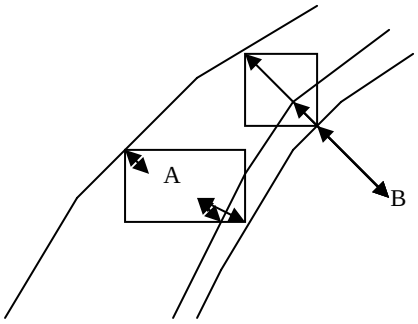

Fig. 3. Computing distance to multiple paths. Point A is inside the box enclosing the three points nearest to it, so its score is 0. Point B is outside the box enclosing the three points nearest to it, so its score is the distance squared to that box.

The test time complexity of multiple path modeling is $O(pkd)$ given p paths, k segments per path, d dimensions and stateless modeling (testing all path segments). Run time improves to $O(pd)$ using sequential testing and maintaining a nearest segment state for each path. Later in Section 4, we will eliminate the $O(p)$ penalty by approximating the training paths with a sequence of boxes that enclose them.

3.1 Path Model Generation

To approximate x with $k - 1$ line segments defined by k vertices, we use a greedy bottom-up approach. The *vertex removal* algorithm removes $n - k$ vertices. Referring to Figure 4, the effect of removing vertex B in the sequence ABC is to replace the two line segments AB and BC with the line segment AC. This induces an error, which we define to be $|AC||BB'|^2$, where $|AC|$ is the length of segment AC, and $|BB'|$ is the distance from B to B', the nearest point on segment AC. The justification for this definition is that if we were to test the training data on itself, then the measured anomaly score would be proportional to our proposed measure, while the true anomaly score should be zero.

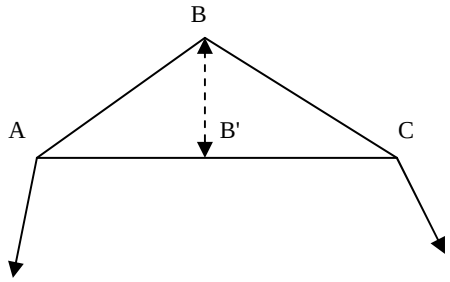

Fig. 4. Removing vertex B induces an error approximated by $|AC||BB'|^2$

An improvement to vertex removal is *path fitting*, in which, after removing B, we shift A and C a distance of $|BB'|/4$ in the direction from B' to B (Fig. 4). If the path is smooth with a gradual curve, then this has the effect of reducing the error because the new segment A'C' is a better fit to ABC than the original AC in the vertex removal algorithm. An optimal shift for AC alone would be $|BB'|/2$, but this would induce too much error in the segments adjacent to A'C'.

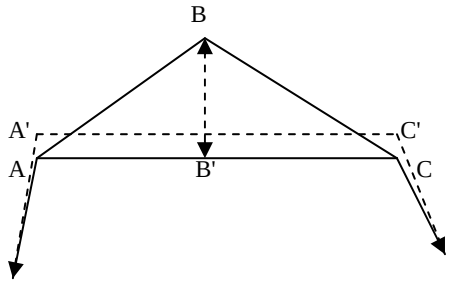

Fig. 5. Path fitting. After removing vertex B, A and C are shifted 1/4 the distance from B' to B to reduce the induced error.

The algorithm for path fitting is given in Fig. 6. The input is the sequence of n vectors $\mathbf{x_i}...\mathbf{x_n}$ in d-dimensional feature space and the desired number of vertices, k. The algorithm runs in $O(n \log n)$ time by storing the vertices in a doubly linked heap, as in the MBR algorithm. The vertices are sorted by error with the smallest at the root. When a vertex is removed, the stored errors of the two nearest neighbors on each side are updated, and they are sifted up or down to restore the heap property. The vertex removal algorithm is identical to path fitting except that the shift is zero

and only one neighbor on each side of the removed vertex needs to be updated.

```
path_fit(x1...xn , k)
  while n > k do
    find i minimizing error(xi)
    b := point on (xi-1,xi+1) nearest xi
    shift := (xi – b)/4
    xi-1 := xi-1 + shift
    xi+1 := xi+1 + shift
    (xi...xn-1) := (xi+1...xn)
    n := n – 1
  return (x1...xk)
```

Fig. 6. Path fitting algorithm

4. BOX MODELING

Building a box model follows the MBR algorithm described in Section 2 with two modifications. First, instead of merging two boxes into one, we merge three boxes into two. Second, we model multiple paths by first constructing a box model of one path, then expanding the boxes to enclose the other paths. In addition, testing differs from MBR in that the test series is not also converted to a box model. This allows us to assign an anomaly score to each test point online.

Box merging is shown in Fig. 7. We first find the box whose removal results in the smallest increase in volume (ignoring overlap between nonadjacent boxes). Then to remove the box, we expand the two neighboring boxes just enough to include the center of the removed box. We call this algorithm MBR3.

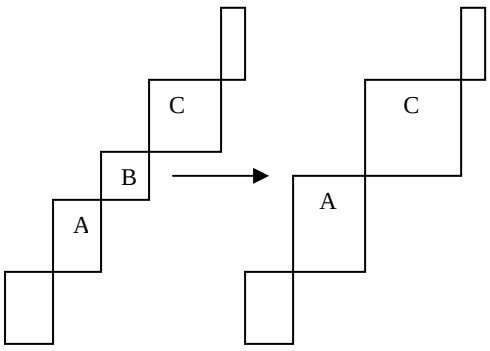

Fig. 7. When box B is removed in MBR3, boxes A and C are grown to enclose the center of B.

The intent of MBR3 is to produce a more uniform distribution of box sizes than MBR. (However we did not test this, nor claim that we succeeded). However MBR3 has the disadvantage that the original path is no longer guaranteed to be enclosed by the new boxes. This occurs when the original path does not pass exactly through the center of the removed box.

The second modification is to expand the k boxes that approximate the first training path to contain all of the remaining p - 1 paths. This is done in two passes for each path. First, we label each point in the path with the box that is closest to it. In the second pass we expand the boxes to enclose the points with matching labels. We do this one path at a time to reduce the space complexity between passes from $O(pk)$ to $O(k)$. Two passes are required because consecutive points in a path tend to be close together, which could result in a pathological model in which a single box grows in small steps to enclose the entire data set. The algorithm is given in Fig. 8..

```
box_expand(x1...xk, y1...yn)
    (x: sequence of k boxes)
    (y: sequence of n points)
    (output: x expanded to enclose y)
  for each yj
    lj = i: xi is closest box to yj
  for each yj
    expand xlj to enclose yi
```

Fig. 8. Expanding box sequence *x* to enclose path *y*.

We recommended that the first path (the input to MBR3) be included in the box expansion step, even if it is the only path. This solves the problem mentioned earlier in which the path may lie slightly outside the box model.

Note that the box model depends on the order in which the paths are presented. We recommend that the most "average" path be used as the initial input to MBR3, and to present the outlier cases last

5. EXPERIMENTAL RESULTS

In this section, we compare path and box modeling with Euclidean distance, DTW and Gecko+RIPPER on the NASA valve data set. The purpose of the experiments is to show that it is possible to construct working anomaly detection systems based on path or box modeling for this data set.

5.1 NASA Valve Data Set

The NASA valve data set [5] consists of solenoid current measurements recorded on Marrotta series MPV-41 valves as they are remotely opened and closed in a laboratory. These small valves are used to actuate larger, hydraulic valves that control the flow of fuel to the space shuttle engines. Sensor readings were recorded using either a shunt resistor or a Hall effect sensor under varying conditions of voltage, temperature, or blockage or forced movement of the poppet to simulate fault conditions.

There are several data subsets, of which two are suitable for testing anomaly detection systems. These are the TEK and VT1 (voltage test 1) sets. The TEK set contains 4 normal and 8 abnormal time series. The four normal traces are labeled TEK 0 through TEK 3, and vary slightly in the degree of background noise, duration of the "on" cycle, and average current during both the "on" and "off" portions. The abnormal series (TEK 10 through 17) were generated by restricting or forcing the movement of the poppet, which has the effect of changing the shape of the rising and falling edges of the waveform. All of the waveforms consist of 1000 samples at a rate of 1 ms per sample. The trace begins at time -0.1s. The valve is actuated at time 0, and deactivated at various times, typically around time 0.2s to 0.3s. The "on" current is approximately 4 in unspecified units.

The "off" current is approximately 0. Measurements are quantized with a resolution of 0.04. In our experiments we do not use TEK 4 through TEK 9 because these are partial waveforms with different sampling rates.

Figure 9 shows three typical waveforms, TEK 0, 10, and 16. TEK 0 is normal. The spikes on the rising and falling edges of the waveform are due to induced voltage caused by movement of the solenoid magnet during opening and closing of the poppet. In TEK 10, the poppet is blocked, so these spikes are absent. In TEK 16, the poppet is initially blocked, then released during the middle of the "on" cycle, causing a temporary dip in the current. It lacks a spike on the rising edge, but has a normal spike on the falling edge.

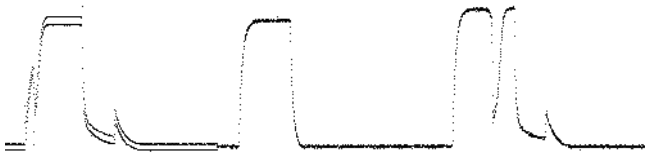

Fig. 9. Concatenation of TEK 0, 10, and 16.

In addition to these differences, there are also differences unrelated to valve failure. TEK 0, 1, and 15 have a 500 Hz signal with amplitude 0.24 as a background signal, visible in the first waveform as a double line. TEK 0 also has a large 2 ms alternating current spike at the start of the falling edge (not visible at this scale) that is absent in the other traces.

The second data set is the VT1 set. This consists of 27 time series recorded under varying conditions of voltage, temperature, and poppet blockage. Each series is 20,000 samples over a period of 2 seconds. In all cases the valve is actuated at time 0.5 sec. and deactivated at time 1.3 sec. For each series there are two readings, the first with a shunt resistor and the second with a Hall effect sensor. In our experiments we use the Hall effect measurement because it is less noisy but otherwise identical. The "off" current is approximately 0 A. The "on" current ranges from 0.42 to 1.08 A, increasing with voltage and decreasing with temperature (due to increased resistance of the solenoid coil). The voltage ranges from 14 V to 32 V in steps of 2 V at room temperature (21C or 22C). At 4 V steps (16, 20, 24, 28, 32) there is an additional recording for high temperature (69C to 71C) and one recording each for a poppet impedance of 4.5 and 9 mils. There are three runs under normal conditions at 32 V, but only one run for all other test conditions. The poppet fails to open at 14 V and at 16 V at high temperature.

In this paper we use the following notation to refer to VT1 traces: *V* for voltage, *T* for high temperature, i45 or i90 for 4.5 or 9.0 mil impedance. For example, *V24i45* denotes 24 V and 4.5 mil impedance. *V32T* denotes 32 V and high temperature.

5.2 Experimental Procedures and Evaluation Criteria

We test each proposed anomaly detection algorithm on the TEK and data sets. In each case we train the model on a proper subset of the training data, assign anomaly scores to all of the traces, and compare the normal and abnormal scores.

We say that an abnormal trace is detected if it has a higher score than all of the normal traces, whether those traces were included in the training set or not. We evaluate an anomaly detection system by the number of detections.

We evaluate the following algorithms.

- Euclidean model (equation (1)), with and without normalization.
- DTW (equation (2)), with and without normalization.
- Gecko with default parameters (tuned to TEK 0-1).
- Path modeling with parameters tuned for best results.
- Box modeling with parameters tuned for best results.

The VT1 set does not label the data as normal or abnormal. In our experiments we define "normal" to be the set of traces at low temperature with no impedance in the range 18 V to 30 V. Thus, there are 7 normal traces: V18, V20, V22, V24, V26, V28 and V30. We use the VT1 set to test the capability of Gecko, path and box modeling to generalize to unseen voltages given a subset of the normal voltages, and to detect temperature and impedance anomalies at unseen voltages. This test arrangement is not suitable for testing Euclidean distance or DTW because they cannot generalize.

By adjusting the threshold on the anomaly scores, different detection and false alarms rates can be obtained. For this study, we choose a threshold that yields no false alarms. That is, the threshold is set to be higher than the anomaly scores obtained from the normal traces (including those that are not used in training). In practice this is reasonable because normal traces are readily available for tuning the threshold and unforeseen bad traces are not available.

5.2.1 Euclidean Distance and DTW

Euclidean modeling requires that the time series be aligned. Recall that only the rising edge of the TEK waveforms are aligned. We test two solutions to the TEK alignment problem.

- Test the rising edge only.
- Manually align the falling edge.

To test the rising edge only, the series are truncated at time 0.1s, at which point the "on" current has stabilized. To align the falling edge, we insert copies of or remove samples at time 0.1s to align the falling edge to 0.2s and then truncate at time 0.78s.

5.2.2 Gecko+RIPPER

We tuned the Gecko parameters to produce the best results we could find on the TEK data set: a consecutive error threshold of 5, a consecutive next state threshold of 1, a smoothing window of size 2, and a derivative window of size 11 (5 before and 5 after). Although a Gecko model can be edited, we did not do so.

Gecko is designed to give a pass/fail result. The test data determines the transitions in a sequential state machine, which either goes to an accepting state or an error state. However, the current version will also produce an anomaly score using a rather complex algorithm which we outline here; see [12] for details. The modification is to run as a "nondeterministic" state machine, in which the state is the set of segments for which the test point satisfies the rules. When a point fails to satisfy the rules of either the current or next segment, that segment is removed from the set. When the set is empty, Gecko goes into a recovery mode in which it tests segments in an exponentially growing window starting at

the last known matching segment. Gecko outputs an anomaly score as a time series which increases by 1 at each step when the set is empty and decreases by 1/3 otherwise. The final score is the sum of these outputs.

5.2.3 Path and Box Modeling

We used the same feature set for path and box modeling. For features, we used the smoothed signal, and the smoothed first and second differences to create a 3-D feature space. We chose the first and second differences because they are intuitive (each test point should match the level, slope, and curvature of a training point), but it is actually the time lag in the smoothing filters that makes the model work. The smoothing is also necessary because the valve data is quite noisy. We selected the filters based largely on visual inspection of the output, and found that additional filtering is needed after each difference operation.

Specifically, we built the filters from two primitive elements, a two tap low pass infinite impulse response filter, *F*, and a two tap finite impulse response difference filter, D. *F* is defined:

$$F(x_i) = \frac{(T-1)F(x_{i-1}) + x_i}{T}$$

where T is the filter time constant and x_i is the input at time i. $F(x_0)$ is initialized to 0. *D* is defined:

$$D(x_i) = x_i - x_{i-1}$$

The three features are:

$$current = F(F(x))$$
$$d\_current = F(F(D(current)))$$
$$d2\_current = F(F(D(d\_current)))$$

To make a distance measure meaningful, each of the features should play a role. In this experiment, we scale the three features to fit a unit cube, so that the training data always ranges from 0 to 1. Other approaches are certainly possible, such as normalizing to unit standard deviation, or specifying the scaling as parameters.

Smoothing allows the output to be subsampled at the rate $1/T$ to speed processing with little loss of information. We do this for all of our experiments.

Figure 10 shows a 3-D view of a path model. Our software allows the user to rotate the image with the mouse, making it easier to visualize. In the figure, the three closely spaced loops are the trajectory path approximations of TEK 0, 2, and 3, each segmented by the path fitting algorithm with k = 25 segments. The outer loop of connected dots is the test path of TEK 16, which has not been approximated. As can be seen, the points on TEK 16 lie far from the three training paths. This model uses a filter time constant of T = 4 ms with subsampling at the same rate.

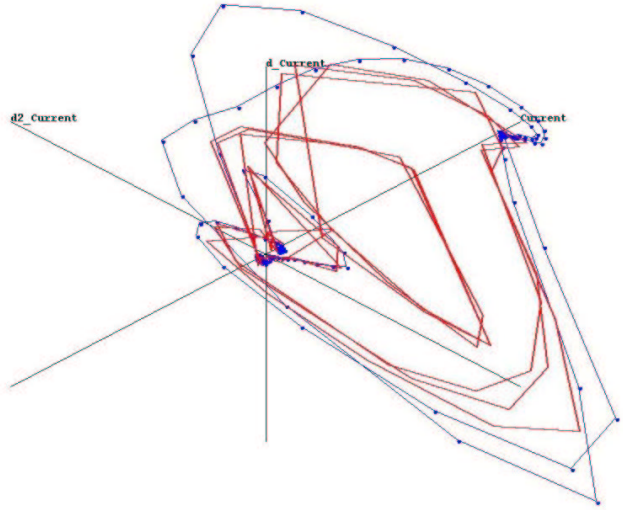

Figure 10. Path model of TEK 0, 2 and 3 with abnormal test path TEK 16.

Figures 11 and 12 shows the equivalent box model with k = 25 boxes. Figure 11 shows a normal test trace, TEK 1, which closely follows the model. Figure 12 shows the same abnormal test trace, TEK 16 as Figure 10, which again deviates from the model.

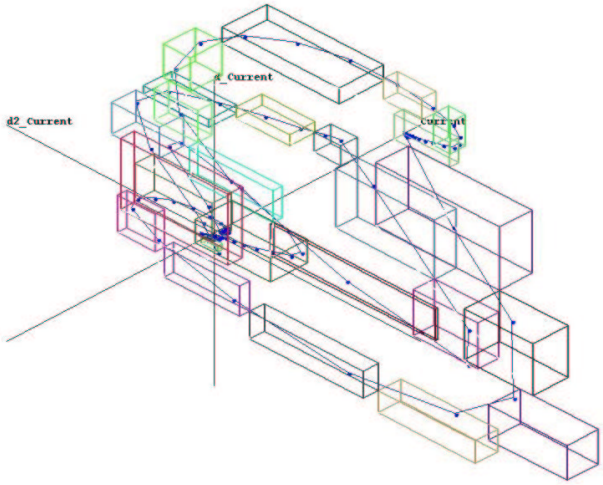

Fig. 11. Box model of TEK 0, 2, 3 with normal test path TEK 1.

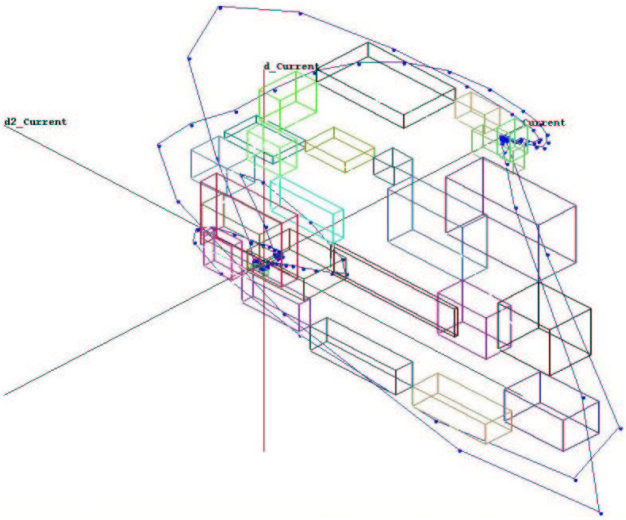

Fig. 12. Box model of TEK 0, 2, 3 with abnormal test path TEK 16.

Path and box modeling allow testing with or without a sequential constraint. The parameter R selects the number of path segments or boxes tested. Segments/boxes are tested in the following order: current, next, previous, second from next, or chosen at random. Thus, $R = 2$ constrains the test data to proceed forward, $R = 3$ allows backwards movement, $R = 5$ allows escape from local minima. $R = k$ tests all boxes or segments. Note that test time complexity is $O(Rpd)$ for path modeling and $O(Rd)$ for box modeling, where p is the number of training paths and d is the number of features.

5.3 TEK Results

For the TEK data set, we label TEK 0 through 3 as normal and TEK 10 through 17 as abnormal. Results are given in Table 1. Recall that an abnormal trace is detected if its score is higher than all of TEK 0-3. The column *Pct 1* gives the percent detected out of the 32 tests with one training trace (8 for each training trace). The column *Pct 2* gives the percent detected out of the 48 tests with two training traces for Gecko and path modeling, or 96 tests for box modeling. The number is higher for box modeling because the training order is significant. N/A means not applicable.

Table 1. TEK test results using 1 or 2 training traces.

Algorithm	**Pct 1**	**Pct 2**
Euclidean, raw, rising edge only	69	N/A
Euclidean, normalized, rising edge only	66	N/A
Euclidean, raw, edited full waveform	69	N/A
DTW, not normalized	41	N/A
DTW, normalized	44	N/A
Gecko + RIPPER	47	65
Path T=5ms, k=25, R=4 or k	100	100
Box, T=5ms, k=20, R=2,3,4,5,k	100	100

Table 2 lists the abnormal traces not detected by each algorithm when trained on one trace.

Table 2. TEK misses with one training trace

Method	TEK 0	TEK 1	TEK 2	TEK 3
Euc raw rise	15,17	14,15	11,14, 15,17	15,17
Euc norm rise	14,15,17	14,17	11,13,15	15,17
Euc raw edit	11,15,17	12,14, 15, 17	11,14,17	14,15, 17
DTW raw	15	15	10-15,17	10-15, 17
DTW norm	15	15	10-15	10-15
Gecko		16	10-17	10-17
Path				
Box				

Table 3 lists the abnormal traces not detected when Gecko is trained on two traces. The results only apply to Gecko because Euclidean distance and DTW allow only one training trace, and because path and box modeling do not miss any anomalies.

Table 3. Gecko misses with two training traces

Training	Missed detections among TEK 10-17
TEK 0, 1	TEK 16
TEK 0, 2	
TEK 0, 3	
TEK 1, 2	
TEK 1, 3	TEK 10-17
TEK 2, 3	TEK 10-17

To be fair, Gecko gives better results (83% detected) when trained on TEK 0, 1, or both, for which it was tuned. The other missed detections are due mainly to a very high false alarm score assigned to TEK 0 when trained on TEK 2 or 3. We did not attempt to tune Gecko for these other training sets.

Path and box modeling generally give good results on the TEK data using a filter time constant of T from about 4 to 10 ms, subsample interval $S \leq T$, $k \geq 25$ path segments or 20 boxes, whether testing with or without sequential constraints.

5.4 VT1 Results

As we mentioned, the VT1 set lacks baselines when used with only one training series, so it is not possible to test Euclidean and DTW on this data set. Instead, we test the generalization capabilities of Gecko, path and box modeling. To do this, we arbitrarily define the range 18 to 30 V, low temperature and no impedance as our normal set. There are 7 traces in this range, allowing us to train on a subset and use the remainder as a baseline. The 20 anomalies consist of low voltage, high voltage, high temperature and impedance.

In this experiment we train on V18, V22, V26 and V30. The order is irrelevant for Gecko and path modeling. For box modeling the training order is V22, V18, V22, V26, V30, following the recommendation of starting in the middle and repeating the first trace (V22). An abnormal trace is counted as

detected if the score is higher than all normal traces including the three normal traces not used in training, V20, V24 and V28. Results are shown in Table 4.

Table 4. VT1 test results.

Algorithm	Pct
Gecko + RIPPER	95% (misses V20i45)
Path, T=5ms, k=20, R=k	100%
Box, T=5ms, k=20, R=k	90% (misses V28T, V32T)

The missed detections by box modeling are higher voltage, high temperature anomalies such as V32T. These are hard to detect because the effects of high voltage and high temperature cancel out to produce a normal looking waveform.

The same range of path and box model parameters that work well on the TEK data also work well on the VT1 data, except that models with a sequential constraint ($R < k$) tend to do poorly.

6. CONCLUSIONS AND FUTURE WORK

We introduced two time series anomaly detection algorithms that that are accurate, not opaque, editable, score each data point (online), efficient, and generalizable from multiple time series. We first extended feature trajectory path models by introducing an efficient but approximate method of testing whether a data point lies between the trained paths. Then we eliminated the test time penalty for multiple paths by extending the MBR model to approximate the set of paths with a sequence of boxes in feature space. A box model is not quite as accurate as a path model, but is faster.

We evaluated our two methods (path and box modeling) against three existing methods (Euclidean, DTW, Gecko) with the shuttle valve data from NASA. For the TEK data, compared to existing algorithms, our methods detected more abnormal traces. For the VT1 data, our methods detected similar or more abnormal time series.

We do not pretend that path or box models are appropriate for all time series. Some work is required to tune parameters to a data set, but this is no different than most other anomaly detection systems. However these models have the nice property that they can be visualized, which should aid in verifying their correctness or modifying them manually to add domain specific knowledge. We did not directly test this capability, however.

In addition to the valve data, path and box modeling have been tested on spring-mass and battery charger simulations with good results. Future work will include online testing to identify anomalous points within a time series, comparison with other algorithms such as CDM, and testing on other data sets, such as arrhythmia detection in ECG traces.

7. ACKNOWLEDGMENTS

This work is supported by NASA (NAS10-02044). Bob Ferrell and Steve Santuro at NASA provided the valve data set. Walter Schefele at ICS developed the visualization software used in this project and provided screenshots for this paper. Stan Salvador and Chris Tanner at Florida Tech. provided test results for Gecko. Eamonn Keogh of UCR provided helpful comments on this paper.

8. REFERENCES

[1] "Inquiry Board Traces Ariane 5 Failure to Overflow Error", SIAM News, 29 (8), October 1996, http://www.siam.org/siamnews/general/ariane.htm

[2] Greg Clark, Alex Canizares, "Navigation Team Was Unfamiliar with Mars Climate Orbiter", space.com, Nov. 10, 1999, http://www.space.com/news/mco_report-b_991110.html

[3] W. Cohen, "Fast Effective Rule Induction", *Proc. ICML*, 1995.

[4] D. Dasgupta and S. Forrest, Artificial Immune Systems in Industrial Applications, *Proc. International Conference on Intelligent Processing and Manufacturing Material (IPMM)*, Honolulu, HI, 1999.

[5] B. Ferrell, S. Santuro, NASA Shuttle Valve Data. http://www.cs.fit.edu/~pkc/nasa/data/ (2005)

[6] Marios Hadjieleftheriou, George Kollios, Vassilis J. Tsotras, Dimitrios Gunopulos, "Efficient Indexing of Spatiotemporal Objects". EDBT 2002: 251-268.

[7] E. Keogh and S. Kasetty, On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration, Proc. *Proc. SIGKDD*, 2002.

[8] E. Keogh, S. Lonardi, C. A. Ratanamahatana, Towards Parameter-Free Data Mining, *Proc. ACM SIGKDD*, 2004.

[9] Richard J. Povinelli, Michael T. Johnson, Andrew C. Lindgren, Jinjin Ye, "Time Series Classification using Gaussian Mixture Models of Reconstructed Phase Spaces," IEEE Transactions on Knowledge and Data Engineering, 16 (6), June 2004, pp. 779-783.

[10] S. Salvador, P. Chan, "FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space", *KDD Workshop on Mining Temporal and Sequential Data*, 2004.

[11] S. Salvador, P. Chan, J. Brodie, Learning States and Rules for Time Series Anomaly Detection, *Proc. 17th Intl. FLAIRS Conf*, pp. 300-305, 2004.

[12] Stan Salvador, "*Learning States for Detecting Anomalies in Time Series*", MS Thesis, Florida Tech, 2004.

[13] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos,. & E. Keogh, "Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures", *Proc. SIGKDD*, 2003

[14] James Watson, "Veteran software makes it to Titan", Personal Computer World, Jan. 26, 2005, http://www.pcw.co.uk/analysis/1160783

[15] A. Ypma, "Learning Methods for Machine Vibration Analysis and Health Monitoring", Dissertation, Delft University of Technology, Netherlands, 2001.