Modeling Multiple Time Series for Anomaly Detection

Philip K. Chan and Matthew V. Mahoney
Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901
pkc, mmahoney@cs.fit.edu

Abstract

Our goal is to generate comprehensible and accurate models from multiple time series for anomaly detection. The models need to produce anomaly scores in an online manner for real-life monitoring tasks. We introduce three algorithms that work in a constructed feature space and evaluate them with a real data set from the NASA shuttle program. Our offline and online evaluations indicate that our algorithms can be more accurate than two existing algorithms.

1. Introduction

Known failures can be modeled and detected, however, unforeseen problems are not defined and hence are much harder to detect. How do humans detect these novel failures? Usually, they rely on their experience on how the system should behave and if deviations from normalcy arise, they investigate the anomalies to determine if they are indeed problems. Similarly, we can endow computers with the capability of detecting anomalies by modeling the normal behavior. Since algorithms do not usually generate the perfect anomaly detector, the ability for humans to comprehend and edit the models for fine-tuning is essential. This is particularly important for anomaly detection algorithms because they usually have high false alarm rates.

In this paper we introduce three algorithms that can construct comprehensible models from *multiple* training time series that are representative of the normal behavior of a system. Instead of storing all the training series, we generalize the series into a concise model, which is then used for anomaly detection. The resulting model is human comprehensible as well as efficient during detection. Rather than working in the time domain inherent in the time series, we construct features and work in a feature space. Model representation is based on a sequence of *axis-parallel* "boxes" in the feature space. The algorithms attempt to find boxes that provide the most specific generalization of the training series. During monitoring, the models produce anomaly scores in an online manner.

Our contributions include algorithms for modeling multiple time series, comprehensible models in a constructed feature space, methodology for offline and online evaluations, improved accuracy over existing methods based on experiments using a real dataset from NASA.

Sec. 2 discusses related work. We describe our algorithms in Sec. 3 and evaluate them in Sec. 4. Sec. 5 summarizes our findings and suggest possible improvements.

2. Related Work

Some proven and broadly applicable techniques such as WCAD [4] 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. Also, WCAD [4] does not generate online anomaly scores. Our work is based on trajectory modeling in feature space as described by Povinelli et. al. [5]. 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. [8] describe a Greedy-Split algorithm that finds minimum bounding rectangles (MBR). It iteratively merges adjacent data points (or rectangles/boxes) in the time series in a bottom-up manner until a user-specified number of boxes (k) is reached. At each iteration, all potential merges are considered and the merge that increases the total volume the least is chosen to be performed. This

greedy algorithm runs in $O(n \log n + k)$ time [8], where n is the length of a time series, and might not yield an optimal solution. However, a dynamic programming algorithm that is optimal requires $O(kn^2)$ [3]. Vlachoes et al's task is to create a database index for efficient querying—given an entire time series, finds the closest one in the database in an offline manner. However, our task is to generate anomaly scores in an online manner—for each data point in the time series, generate a score for detecting novel anomalies.

In our earlier work on the NASA valve data [2], we used the Gecko and RIPPER algorithms [7] to create a bounded rectangle model. The Gecko clustering/segmentation algorithm 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 a 3-D feature space: the original signal, and the first and second derivatives, each of which is smoothed by a low-pass filter. The time series is then segmented in feature space using a hierarchical clustering algorithm. Next, RIPPER [1] is used to generate a minimal rule set which separates the segments. Each rule represents a "box" in the feature space. One segment can be defined by several boxes, and some boxes may be open on some sides. Gecko, like MBR, satisfies our criteria that the model be comprehensible. The boxes 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 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. 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. Approach

Our main goal is to generate a comprehensible model that characterizes the behavior of multiple normal time series and accurately detects time series signifying unacceptable behavior as anomalies. The normal training time series are assumed to have similar general behavior, but they may not be aligned—segments in two time series with corresponding behavior might not begin and end at the same time in the time series. Also, given a test time series and a model, anomaly scores are generated in an online manner (applicable for real-life anomaly detection). The model consists of axis-parallel constraints ("boxes") in the feature space—each feature is constrained between a minimum and a maximum value. We first discuss how a model is generated for one time series, then for multiple time series.

```
Greedy-Split(x, k)
  for i = 1 to n-1
    x[i] = merge(x[i], x[i+1])
  delete x[n]
  while n > k
    find i minimizing
      Vdiff = V(i, i+1) - V(i) - V(i+1)
    x[i] = merge(x[i], x[i+1])
    delete x[i+1]
  return x
```

Figure 1. Greedy-Split Algorithm

We use the Greedy-Split algorithm [8] to generate a model for one time series. A sequence of n points in a 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. That is, Greedy-Split attempts to find a sequence of k boxes with the least total volume and hence a model that is the most specific generalization of the original sequence of data points. The pseudocode for Greedy-Split algorithm that models a sequence `x` of length `n` with `k` boxes is in Fig. 1. In the Greedy-Split algorithm, `merge(a, b)` replaces points or boxes `a` and `b` with the smallest box that encloses both, `V(i)` is the volume of `x[i]`, and `V(i, i+1)` is the volume of `merge(x[i], x[i+1])`. `Vdiff` is the increase in volume that would result from merging. Deleting an element `x[i]` implicitly decrements `n`.

Greedy-Split can run in $O(n \log n)$ time by storing the boxes in a heap (priority queue) ordered by `Vdiff`, 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 `Vdiff` 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.

3.1 Box Modeling for Multiple Training Series

To generate a model for s time series, we first uses Greedy-Split to find k boxes in one time series and expand the boxes to contain all of the remaining $s-1$ training time series, processing one series at a time. This is performed in two passes for each time series. First, we label each point in the time series with the box that is closest to it. In the second

```
boxExpand(x, y)
  for j = 1 to n
    l[j] = i where x[i] is closest to y[j]
  for j = 1 to n
    expand x[l[j]] to enclose y[i]
  return x
```

Figure 2. Expanding box sequence x to enclose time series y.

```
1. for each training series
     use Greedy-Split with k1 boxes
2. for each box j in each training series i
     for each training series k != i
       label box j with nearest box in k
3. for each box in all series
     merge with the labeled nearest boxes in
       the other series
4. while there are more than k boxes
     find the two closest boxes and merge them
```

Figure 3. Order-independent Box Modeling

pass we expand the boxes to enclose the points with matching labels. We perform this step one time series at a time to reduce the space complexity between passes from $O(sk)$ to $O(k)$. Two passes are required because consecutive points in a time series 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 pseudocode of the algorithm is in Fig. 2. `x` is a sequence of k boxes and `y` is a sequence of n points. Note that `boxExpand` is called for each time series and the order of the time series affects the resulting box model that covers all s series. That "order-dependent" effect might not be desirable. Hence, we explore an "order-independent" algorithm next.

3.2 Order-Independent Box Modeling

The main idea is to first approximate each time series by boxes and then select boxes that are closest to merge/expand. During the merging, we ensure that each merged box contains at least one box from each time series. That is, we bias the model such that each merged box is generalized over all training series. The pseudocode is illustrated in Fig. 3. Step 1 generates boxes for each individual training series. Step 2, for each box in each series, finds the closest box from each of the other series. "Closest" means the least increase in volume that would result from replacing a set of boxes with the merged box which would enclose them. This distance can be negative if the boxes overlap. For each box in each series, Step 3 merges the boxes founded in Step 2. While the total number of boxes exceeds k, Step 4 repeatedly merges the two closest boxes. Steps 2 and 3 ensure that each resulting box covers at least one data point in each series; Step 4 preserves this characteristic. The dominating step is Step 2 or 3, which is $O(k1^3)$, where $k1$ is the initial number of boxes. The algorithm also works without Step 1, but Step 2 or 3 will become $O(n^3)$, where n is the length of a time series.

```
1. for each training series
     use Greedy-Split with k1 boxes
2. while there are more than k boxes
     find the two closest boxes and merge them
```

Figure 4. Order-independent Box Modeling without the All-series Constraint

3.3 Order-Independent Modeling without the All-series Constraint

The order-independent modeling algorithm in the previous section ensures all resulting boxes cover some data points on every training series—we call this constraint (generalization bias) the "All-series" constraint. However, this constraint might not be appropriate. Considering only one feature and four boxes that represent the same behavioral segment in four time series. The four boxes (or ranges in one dimension/feature) are [0,2], [1,3], [10,12], and [11,13]. With the "All-series" constraint, one range, [0,13], will be formed. Naturally, it might be more appropriate to have two ranges: [0,3] and [11,13], which might not be possible with the "all-series" constraint. To remove the constraint and allow "parallel" boxes, we remove Steps 2 and 3 that are for enforcing the constraint in Fig. 3. The pseudocode for order-independent modeling without the All-series constraint is in Fig. 4. Allowing "parallel" boxes can reduce the increase in total volume and hence generalization.

The two algorithms for order-independent box modeling does not depend on the order of the training series, but the original order of the boxes might not be preserved. Considering only two time series, Box *a* is before Box *b* (not necessarily adjacent) in Series 1 and Box *x* is before Box *y* in Series 2. If *a* is closer to *y* than *x*, *a* and *y* are merged into Box *ay*. If *b* is closer to *x* than *y*, *b* and *x* are merged into Box *bx*. If we want to maintain the ordering of *a* before *b* and *x* before *y*, an ordering is not possible for *ay* and *bx*. Hence, the merged boxes might not have a total ordering consistent with the ordering prior to merging.

3.4. Anomaly Score during Testing

During testing each data point of a time series is assigned an anomaly score in an online manner. If the boxes in a box model has an ordering, we can use stateful or stateless testing. Otherwise, only stateless testing may be performed. The order-dependent algorithm provides an ordering of the

resulting boxes, but the two order-independent algorithms may not provide an ordering. In stateful testing, we keep track of which state/box we were in previously. If the current data point fits the current or next state/box, the anomaly score is zero. Otherwise, the anomaly score is the squared distance to the surface of the current or next box, whichever is closer. In stateless testing all boxes/states are compared. If the data point is within a box, the anomaly score is zero. Otherwise, the closest box is located and the squared distance to the box surface is the anomaly score. Formally, given a point p at (x, y, z) in a 3-D space and a box b at $([x_{min}, x_{max}], [y_{min}, y_{max}], [z_{min}, z_{max}])$. The center of b is at $(\bar{x}, \bar{y}, \bar{z})$, where $\bar{x} = (x_{min} + x_{max})/2$ and similarly for $\bar{y}$ and $\bar{z}$). To find the closest box, we calculate:

$$Distance(p, b) = \sqrt{(x - \bar{x})^2 + (y - \bar{y})^2 + (z - \bar{z})^2}. \quad (1)$$

If point p is inside box b ($x_{min} \leq x \leq x_{max}$ and similarly for y and z), $AnomalyScore(p, b)$ is 0. Otherwise,

$$AnomalyScore(p, b) = (x-\hat{x})^2+(y-\hat{y})^2+(z-\hat{z})^2, \quad (2)$$

where ($\hat{x}$, $\hat{y}$, $\hat{z}$) is the point on the box surface that is closest to p; $\hat{x}$ is defined as:

$$\hat{x} = \begin{cases} x_{min} & \text{if } x < x_{min} \\ x_{max} & \text{if } x > x_{max} \\ x & \text{otherwise} \end{cases}$$

and $\hat{y}$ and $\hat{z}$ are similarly defined. For example, if the range of the x coordinates for box b is $[x_{min} = 3, x_{max} = 5]$ and x is 7, $\hat{x}$ is 5; if x is 4, $\hat{x}$ is 4.

4. Experimental Evaluation

4.1. Experimental Data

The NASA valve data set [2] 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.

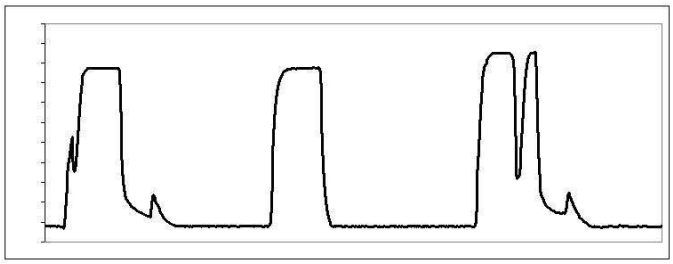

Figure 5. Concatenation of TEK 0, 10, and 16.

Fig. 5 shows three typical series, 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.

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 (invisible at this scale) that is absent in the other traces.

4.2. Experimental Procedures

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.

- Euclidean Modeling (method "EU")
- Gecko+RIPPER (method "GR")
- Order-dependent Box modeling (method "OD")
- Order-independent Box Modeling with the All-series Constraint (method "OC")
- Order-independent Box Modeling without the All-series Constraint (method "OU")

4.2.1 Euclidean (EU)

The Euclidean distance between time series A and B of equal length can be defined as:

$$Euclidean(A, B) = \sqrt{\sum_i (A[i] - B[i])^2}. \quad (3)$$

Since Eucldiean distance is not defined between one (test) time series and a set of (training) time series, we use a nearest neighbor method to compute an anomaly score for each point on the test series. That is, the anomaly score of a test point is the distance to the closest train point with the same index (i). That is, all the training points are stored and no generalization occurs during training. Since Euclidean inherently does not perform alignment, we perform a simple alignment before calculating the Euclidean distance. Given a test series, we shift each entire training series left or right so that the rising edges are aligned ("extra" data points are "wrapped around"). We also generate another training series so that the falling edges are aligned. That is, each training series generates two shifted series with two different alignments and the number of training series is doubled. To save computation, we use the square of the Euclidean distance as the anomaly score.

4.2.2 Gecko+RIPPER (GR)

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). 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 an algorithm which we outline here (see [7] 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.

4.2.3 Box Modeling (OD, OC, OU)

We used the same feature set for path and box modeling. For features, we used the smoothed signal, and the

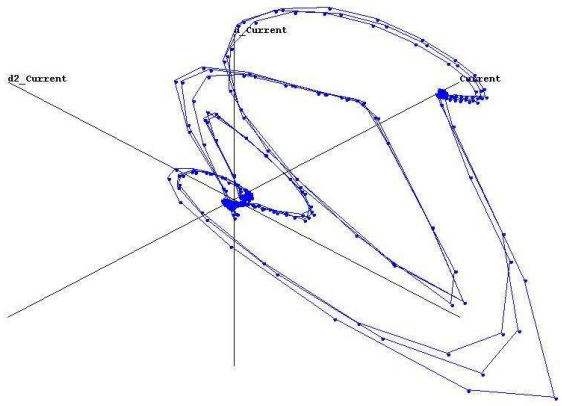

Figure 6. Tek 0, 1, and 3 in our feature space.

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:

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

Fig. 6 shows TEK 0, 1, and 3 in this feature space: $current$, $d\_current$, and $d2\_current$. Note that time is not a feature.

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.

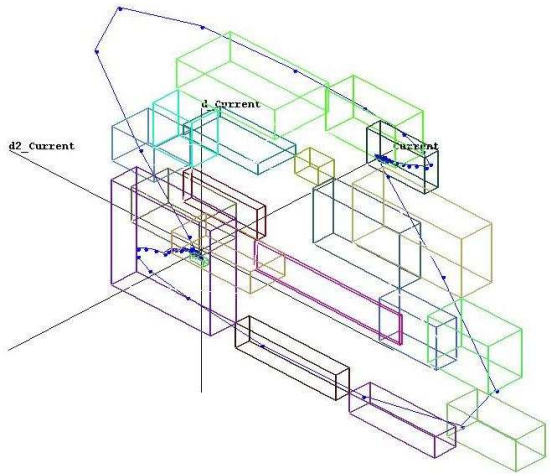

Figure 7. TEK10 tested with the OD model

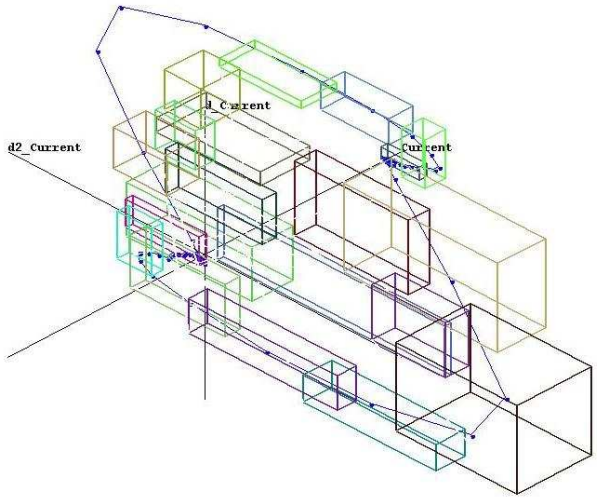

Figure 8. TEK10 tested with the OC model

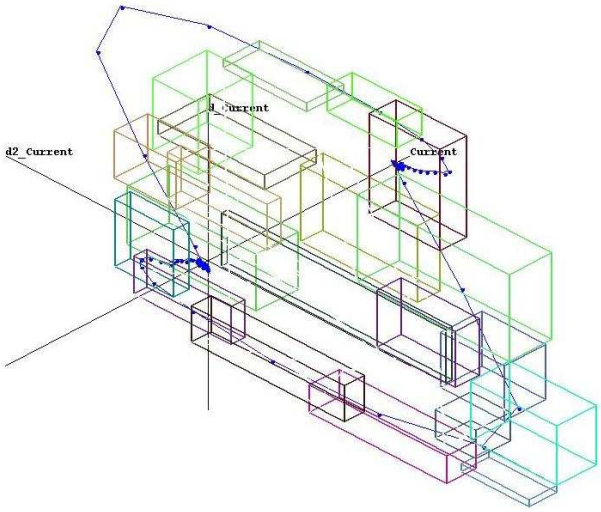

Figure 9. TEK10 tested with the OU model

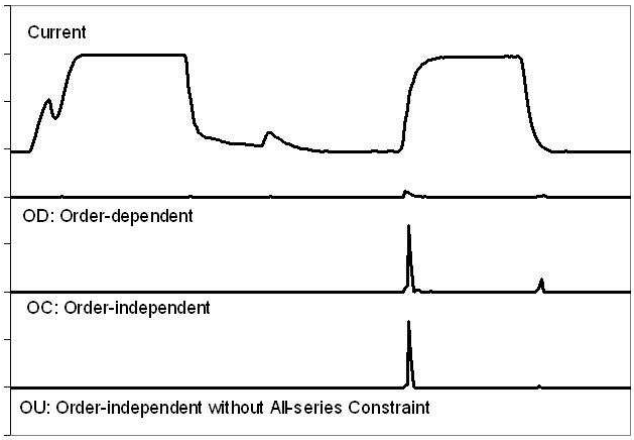

Figure 10. Anomaly scores for TEK 2 and 10

Figures 7, 8, and 9 show the resulting box models using Methods OD, OC, and OU (Sec. 4.2) and TEK 0, 1, and 3 as training series (Fig. 6) with K=20 boxes. Note that the training series diverse quite a bit in the lower right corner of Fig. 6. As we expect, the OC model (with the All-series constraint) in Fig. 8 has one box in the lower right corner while the OU model (without the All-series constraint) in Fig. 9 has multiple boxes. An abnormal series TEK 10 is shown with the box models to illustrate its deviation from the learned box models.

In Fig. 10 we show the anomaly scores of TEK 2, which is normal, and TEK 10, which is abnormal. As we observe from the figures, TEK 2 has zero/low anomaly scores, while TEK 10 has high anomaly scores near the rising and falling edges of the time series. For our experiments, we use stateless testing (Sec. 3.4), time delay T=5, initial boxes K1=200, and resulting boxes K=20. These parameters were selected based on some initial experiments.

4.3. Offline Evaluation

We first evaluate the total anomaly score for an entire test time series in an offline manner (assuming the entire time series is available before testing starts and prediction occurs after the entire time series has been processed). Each time series is labeled either normal or abnormal. By adjusting the threshold on the total anomaly scores, different detection and false alarms rates can be obtained. 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 series (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 abnormal series are not available. Table 1 shows the number of misses (missed detections) from models with two and three training time series. The columns indicate which training series

Table 1. Number of Misses with No False Alarms with (a) 2 and (b) 3 Training Series

Methods	0,1	0,2	0,3	1,2	1,3	2,3	Avg
EU	6	6	1	5	3	3	4.0
GR	1	0	0	0	8	8	2.8
OD	0	0	0	0	0	0	0
OC	0	0	0	0	0	0	0
OU	0	0	0	0	0	0	0

Methods	0,1,2	0,1,3	0,2,3	1,2,3	Avg
EU	6	1	1	3	2.75
GR	0	0	0	1	0.25
OD	0	0	0	0	0
OC	0	0	0	0	0
OU	0	0	0	0	0

were used. For example, column "0,1" means TEK 0 and 1 are the training series, and "0,1,2" means TEK 0, 1 and 2 are the train series. The "Avg" column is the average of the different training combinations. The same column format is used for tables in subsequent sections.

We observe that box modeling (OD, OC, and OU) have no missed detections, while EU has the most. Both EU and GR benefit from more training data. Different combinations of training series can yield vastly different results. For example, GR has no misses with three of the four combinations of three series, but it missed all 8 abnormal series with two of the six combinations of two series.

4.4. Online Evaluation with ROC

In practice, data points in a time series stream in an online manner and the operator needs to identify problems at the earliest. Hence, instead of evaluating the total anomaly score of each test time series, we would like to evaluate the individual score for each data point. Given the segments that are labeled abnormal (instead of the entire time series), we use a score threshold and check if the anomaly score of each test data point can predict the abnormal segments correctly. For this evaluation, we choose to use the area under the ROC (Receiver Operating Characteristics) curve [6] as the measure. The area corresponds to the likelihood of detection with any threshold. For each model, we evaluate anomaly scores for all 4 normal TEK and 8 abnormal time series. Table 2 shows the area under the ROC obtained from models with two and three training time series.

We observe that GR and EU achieved a larger area than the box modeling methods. Among the box modeling methods, OD (order-dependent) yielded larger area than OC and OU (order-independent). Again, GR benefited from using more training data. While GR and EU are more accurate than box modeling according to the online ROC area evaluation, they are less accurate according to the offline number of detections (Table 1). Since the area under the ROC curve consider all thresholds, all false alarm rates are equally important. Though this evaluation is useful for an arbitrary threshold (false-alarm rate), it might be less applicable to monitoring tasks that try to detect problems.

4.5. Online Evaluation with ROC up to 1% False Alarm Rate

If the false alarm rate is high, the human operator who monitors the device might ignore the alarms, which could render the detection system useless. Also, anomaly detection is prone to high false alarm rates. Hence, we would like to evaluate the performance of the methods at low false alarm rates and measure the area under the ROC curve up to 1% false alarm rate (instead of up to 100% false alarm rate—the entire ROC curve). That is, we measure the performance of the methods with any threshold that causes at most 1% false alarms. Table 3 shows the area under the ROC up to 1% false alarm rate obtained from models with two and three training time series.

From Table 3a, we observe that box modeling (OD, OC, OU) is more accurate than GR and EU. GR, again, benefited from more training series and yielded higher accuracy with three training series instead of two. With three training series (Table 3b) GR's accuracy is similar to box modeling, however the area for GR varies quite a bit (larger variance). The low accuracy for "1,2,3" coincides with the only missed detection observed for GR in Table 1b.

With two training series in Table 3a, box modeling outperformed GR and EU, but the converse is true if the area under the entire ROC curve is considered (Table 2a). This implies that, EU and GR are less accurate below 1% false alarm rate, but more accurate at higher false alarm rates. Box modeling (OD, OC, OU) continues to be insensitive to the amount of training data. This implies that fewer training series are needed and training time could be shorter to achieve similar top accuracy in this study.

5. Concluding Remarks

We introduced three box modeling algorithms that can produce comprehensible models from multiple training time series. The resulting box models can be visualized and edited (software that can rotate and edit the box models in 3-D is not described here). Our first algorithm generates boxes using the Greedy-Split algorithm and then expands them to cover points on the other time series, one at a time. Since the ordering of the training series affects the resulting box

Table 2. ROC Area with (a) 2 and (b) 3 Training Series

Methods	0,1	0,2	0,3	1,2	1,3	2,3	Avg
EU	.77	.84	.85	.85	.88	.85	.84
GR	.70	.90	.90	.87	.83	.86	.84
OD	.68	.72	.72	.72	.72	.72	.71
OC	.64	.66	.62	.66	.66	.68	.65
OU	.64	.67	.61	.66	.66	.68	.65

Methods	0,1,2	0,1,3	0,2,3	1,2,3	Avg
EU	.85	.86	.86	.88	.86
GR	.92	.92	.91	.91	.92
OD	.72	.72	.72	.73	.72
OC	.67	.66	.67	.67	.67
OU	.67	.66	.67	.67	.67

Table 3. ROC Area ($\times 10^{-2}$) upto 1% FA with (a) 2 and (b) 3 Training Series

Methods	0,1	0,2	0,3	1,2	1,3	2,3	Avg
EU	.01	.01	.07	.03	.06	.05	.04
GR	.03	.17	.18	.05	.00	.00	.07
OD	.09	.13	.13	.15	.15	.13	.13
OC	.09	.13	.14	.13	.14	.15	.13
OU	.09	.14	.13	.14	.15	.15	.13

Methods	0,1,2	0,1,3	0,2,3	1,2,3	Avg
EU	.01	.07	.07	.06	.05
GR	.18	.20	.15	.05	.15
OD	.13	.13	.13	.15	.14
OC	.13	.13	.12	.14	.13
OU	.14	.14	.13	.14	.14

model, we introduce an order-independent algorithm. The second algorithm generates boxes on each training series and then merges the boxes with the constraint that each resulting box contains at least one data point on each training series. However, this "all-series" constraint might merge boxes that are too distant so we remove the constraint in our third algorithm and allow less increase in total volume of the resulting box model.

We used the NASA valve data set to compare our box modeling algorithms with Euclidean (EU) and Gecko+Ripper (GR). Our empirical results indicate that our box modeling algorithms have fewer missed detections than GR or EU in an offline evaluation. For online evaluation, our results indicate that box modeling is more accurate than GR or EU at low false rates ($< 1\%$) with two training series and are similar to GR with three training series. Furthermore, the accuracy of box modeling at low false alarm rates with two training series is comparable to the accuracy of GR with three training series.

In the (training) order-independent algorithms proximity among boxes is the main criterion for merging them. However, the original temporal ordering of the boxes in each of the training series is not considered. If we consider the original temporal ordering, we can ignore possible merges that violate the ordering and hence speed up training. Also, boxes in our models are axis-parallel, we can allow more flexible non-axis-parallel boxes with the price of increase complexity. One caveat with online evaluation is the difficulty of labeling the abnormal time series. Do we only label segments that indicate the beginning of abnormal behavior, or do we also label subsequent segments as well and how long these subsequent segments last? We currently use binary (normal/abnormal) labeling, but multi-valued labeling can be explored for different kinds of abnormal segments.

Acknowledgments This work is partially 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 screen shots for this paper. Chris Tanner at Florida Tech. provided test results for Gecko+RIPPER. We thank the anonymous reviewers for their comments.

References

[1] W. Cohen. Fast effective rule induction. In *Proc. 12th Intl. Conf. Machine Learning*, pages 115–123, 1995.

[2] B. Ferrell and S. Santuro. Nasa shuttle valve data. http://www.cs.fit.edu/˜pkc/nasa/data/, 2005.

[3] M. Hadjieleftheriou, G. Kollios, V. Tsotras, and D. Gunopulos. Efficient indexing of spatiotemporal objects. In *Proc. EDBT*, pages 251–268, 2002.

[4] E. Keogh, S. Lonardi, and C. Ratanamahatana. Towards parameter-free data mining. In *Proc. ACM SIGKDD*, pages 206–215, 2004.

[5] R. Povinelli, M. Johnson, A. Lindgren, and J. Ye. Time series classification using gaussian mixture models of reconstructed phase spaces. *IEEE Trans. Knowledge and Data Engineering*, 16(6):779–783, 2004.

[6] F. Provost and T. Fawcett. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *Proc. 3rd Intl. Conf. Knowledge Discovery and Data Mining*, pages 43–48, 1997.

[7] S. Salvador and P. Chan. Learning states and rules for detecting anomalies in time series. *Applied Intelligence*, 2005. to appear.

[8] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *Proc. ACM SIGKDD*, pages 216–225, 2003.