AURA-Alert: The use of Binary Associative Memories for Condition Monitoring Applications

Jim Austin[+,*], Grant Brewer[*], Tom Jackson[+], Victoria J. Hodge[+]

+Department of Computer Science, University of York
York, YO10 5DD, UK
+44 (0)1904 433382
+44 (0)1904 432767
[austin, tom.jackson, vicky]@cs.york.ac.uk

*Cybula Ltd.
IT Centre, Heslington
York, YO10 5DG, UK
grant@cybula.com, austin@cybula.com

Abstract
Many Condition Monitoring (CM) domains are suffering from the dual challenges of substantial increases in the volumes of data being produced and collected by sensing systems, and the challenges of modelling increasing complexity in the remote monitored systems. These two issues give rise to the problem that fast and reliable data mining of CM data is a computationally demanding task for real-time (or near real-time) applications. We present the use of AURA [1], a class of binary associative network built on correlation matrix memories (CMMs), as an underpinning technology for efficient, scalable pattern recognition in complex and large scale CM applications.

AURA is a class of binary neural network. However, it has a number of advantages over standard neural network techniques for CM pattern classification tasks. These include; high levels of data compression, one-pass training for on-line training, a scalable architecture that can be readily mapped onto high performance computing platforms, and a sound theoretical basis to determine the bounds of the system operation. We describe applications illustrating how the AURA system can be optimised to create an extremely efficient and scalable k-nearest neighbour classifier for multi-variate models. We will also illustrate how the one-pass training capability of the AURA system can be used as the basis of normality and exception modelling in complex CM systems. This latter application has particularly powerful advantages for fault detection models in domains which are characterised by highly dynamic trends or drifting in the standard operational mode of a system, and which, as a result, are extremely difficult to accurately model. The application of the AURA techniques will be illustrated with industry led exemplars in the transport and energy sectors.

1. Introduction

Many Condition Monitoring (CM) domains are suffering from the dual challenges of substantial increases in the volumes of data being produced by sensing systems, and the challenges of modelling increasing complexity in the remote monitored systems. These two issues give rise to the problem that fast and reliable data mining of CM data is a computationally demanding task for real-time (or near real-time) applications.

We present the use of AURA [1, 2], a class of binary associative network built on correlation matrix memories (CMMs), as an underpinning technology for efficient,

scalable pattern recognition in complex and large scale CM applications. AURA has a number of advantages over standard neural network techniques for CM pattern classification tasks. These include; high levels of data compression, one-pass on-line training, a scalable architecture that can be readily mapped onto high performance computing platforms, and a sound theoretical basis to determine the bounds of the system operation. We describe applications illustrating how the AURA system can be optimised to create an extremely efficient and scalable k-nearest neighbour classifier for multivariate models. We also illustrate how the one-pass training capability of AURA can be used as the basis of normality modelling in complex CM systems. This has powerful advantages for fault detection models in domains characterised by highly dynamic trends or drifting in the standard operational mode of a system, and which, as a result, are extremely difficult to accurately model. The application of the AURA techniques will be illustrated with industry led exemplars in the transport and energy sectors.

2. The Problem Domain

The focus of the paper is on real-time condition monitoring applications, where it is critical to identify deviations from normal behaviour in sensor data. This applies to a large class of CM application areas, and we will illustrate the concepts drawing from examples in power industry and transport management.

A key element of condition monitoring is the early detection of potential faults in the monitored system or asset, allowing preventative action to be taken before major damage occurs. The condition monitoring system has to identify these potential faults based on the values of a large number of variables, which can either be direct sensor readings or calculated values derived from these. Depending on the potential fault that is occurring, there may be different clues in the data that allow the problem to be identified. The clues that are available can take one of three different forms and each provides different challenges to the asset monitoring system and requires a different approach.

1) The easiest faults to identify just require the thresholding of individual variables. If a sensor reading or derived value exceeds a threshold value then it is known that a certain problem has occurred. Alternatively, there may be a value range during normal operation and any readings not within this range may indicate abnormal activity.
2) Some faults may only be detectable by identifying coincident patterns or value shifts amongst multiple variables, with no individual variable departing from its expected operational range. There are many pattern recognition techniques that can be applied in these scenarios, but all require either example data from previous occurrences of these faults or a degree of expert knowledge.
3) A more challenging category of fault relates to those problems that have not been foreseen during the asset development, have not happened before and do not involve any one variable departing from its normal operating range. The pattern recognition techniques used for type 2 cannot be applied, as there are no examples of the fault to provide training data. Therefore, to detect such faults the asset monitoring system must store a representation of normal behaviour and issue warnings when the asset's activity deviates from this behaviour.

The detection system discussed in this paper is called **AURA-Alert** and focuses on this third scenario, although it would be possible to develop the system so that it can also be

used to identify type 2 faults. AURA-Alert is provided with the data from an extended period of time during which the asset has been known to perform correctly. At regular time intervals during this period, the values of a representative set of variables from the data are converted into a pattern, which represents the state of the asset at that time instance. This pattern is then stored in an AURA associative memory. During asset operation, the current state of the system can be compared to the stored normal operating behaviour to see if that combination of variable values has been seen previously. If the combination has not been seen before, this could be indicative of a problem, even if no individual variables have deviated from their normal value range.

3. State Transitions

At any period in time, an asset with N sensors can be considered to be in a particular state within an N-dimensional space. This can be most easily depicted in two dimensions as shown in the graph in Figure 1. The values from the two sensors point to a unique position within the graph and these are recorded as dots within the state space, whilst the lines linking the dots show how the state of the asset moves around over time. During learning, all the locations within the state space that are visited by the asset, during a period of normal asset activity, are recorded.

During recall, the current values of the two variables will also correspond to a point within the state space, as shown by the red dot in Figure 1. If this red dot is on top of one of the stored blue dots, then the asset is currently in a state that has been seen before during normal behaviour. However, if the red dot is not on top of any of the previously stored states, the system is behaving differently from before and the greater the distance, the more unusual the current state.

Further information can also be obtained from the proposed asset monitoring system. Firstly, it can work out how often the asset has been in the current state, by studying the number of stored states that are close to the recalled point. Also, by labelling the stored states with information about the asset at the time, further information can be obtained about when the asset was last in this state. For example, a previously seen fault could be stored and labelled as an error state, providing a solution to the second of the fault detection categories discussed earlier.

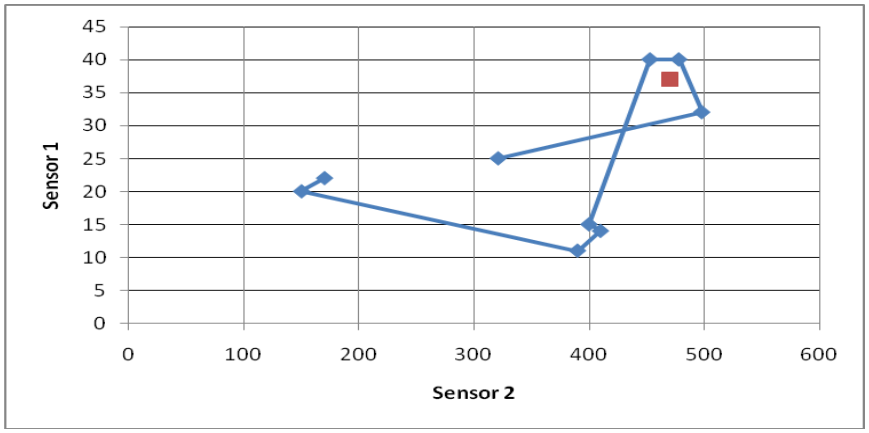


Figure 1 : Transitions in a two-dimensional state space

Although this system can be depicted most easily in two dimensions, most asset monitoring systems will need to store the values of many more than two sensor readings. When a third variable is added, the state of the system will move around in a cubic space and beyond this, N dimensions will create an N-dimensional

hyperrectangle, in which the asset states will lie. Although this is harder to visualise, the system works in exactly the same way, with the locations of historical asset states recorded and then compared to the current state.

One approach to condition monitoring in these domains is the use of a nearest neighbour classifier, which could be implemented by storing the raw values of each variable at each time interval and comparing the recalled points to each of these. However, the resultant time complexity of performing these separate comparisons severely limits the number of states that can be stored in the system. The AURA neural network can search millions of states very quickly [2]. In previous work it has been shown that AURA k-NN performs up to four times faster than the traditional k-NN [3].

4. System Overview

An overview of the training process can be seen in Figure 2a. Firstly, a binning algorithm is used to convert the value of each variable into a binary code, in which only one bit is set. The codes for each variable are then concatenated to create a binary representation of the state, which is stored in a binary Correlation Matrix Memory (CMM) [4] of an AURA associative memory [1]. A binary CMM is a single layer, fully connected network that is capable of very fast storage and retrieval of data.

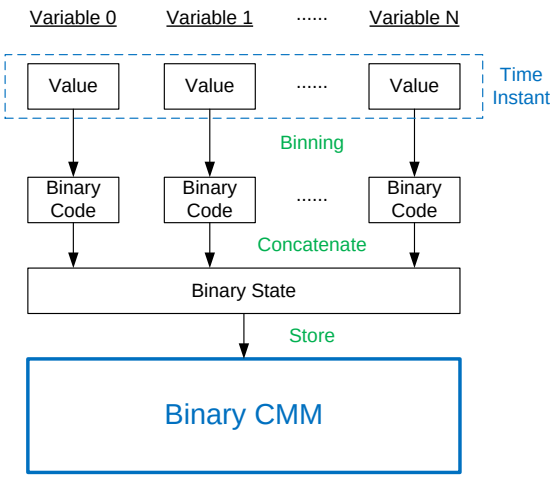


Figure 2a : An Overview of the AURA Alert Learning Process

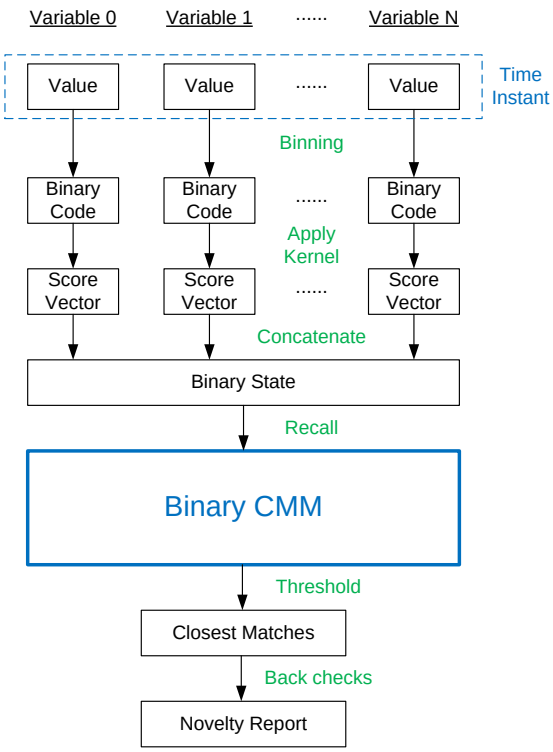


Figure 2b : An Overview of the AURA Alert Recalling Process

The recall process is similar and is summarised in Figure 2b. Again, binning is used to convert each value into a binary code, but these are then spread using a kernel to create a vector of scores. The scores decrease in value as the distance from the set bit increases. This score vector is then presented to the binary CMM and the output thresholded to reveal the closest stored matches to the input pattern. The closest matching time instances are then compared to the original recall values to determine whether a novel operating state has been detected.

Converting a set of variables into a binary pattern

To store a state, the set of variable values must be first converted into a binary pattern which can be stored in the AURA memory. This requires a binning process, with each potential value for each variable assigned to a bin and each bin corresponding to a different bit that will be set in the binary pattern. This process is depicted in Figure 3. Once each variable from a state has been assigned to a bin, the binary pattern representing that state can be created by concatenating the binary patterns together.

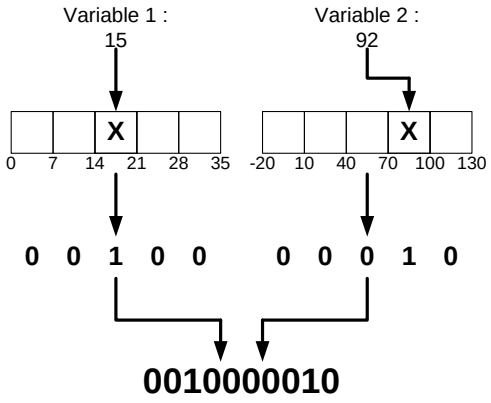


Figure 3 : Converting a state set into a binary pattern

Binning

The number of bins required for each variable and how the values are assigned to the bins are important operational parameters that significantly affect the behaviour of the system. The simplest binning method is to use bins of equal width, as shown in Figure 3, with the full range of potential values split evenly between the bins available. For example, if a variable can take values from 0 to 100 and there are 10 bins available, values between 0 and 10 will be assigned to the first bin, 10 and 20 to the second bin and so on.

One problem with the even width binning system is that the full variable range may only be used in unusual circumstances, stretching each bin unnecessarily and causing detail to be lost where it is required. For example, a temperature sensor may record values of between 200 and 220 degrees when a machine is operational, but only 20 degrees when it is switched off. Using even width bins over the full range of values will therefore result in the majority of values being assigned to the same few bins.

One possible solution to this problem is to use the bins to cover only the important range of values and either discard the time instances where the value does not fall within this range, or place them in the outer bins. Alternatively, fixed frequency binning can be used, where the range of each bin is scaled so that there is an equal number of time instances placed in each bin [5]. This will increase the sensitivity of the binning system in the regions of the state space that are most densely populated. Whether this is advantageous depends on whether it is small changes in the populated regions, or larger changes in the less frequented areas, which are more indicative of a change in the behaviour of the system.

Another design issue that must be considered is that the full range of values for a variable may be unknown when the asset monitoring system is set up. In such instances, the training data can be used to set the upper and lower bounds of the binning range, but there may still be instances of values encountered during recall that have not

been seen during training. Such an occurrence may be significant enough to flag a potential error without even considering the other variable states. One option for AURA approach is to set a threshold for the number of values that can be placed in the extreme bins and, once exceeded, the bins values are reset and the memory retrained [5].

Storing the binary patterns

Using the above method for the conversion of the state at each time interval to a binary pattern, a period of normal behaviour can be stored in the binary CMM of an AURA memory [1]. The CMM is initialised with all weights having strength 0 and associations are stored by placing each input vector along one side of the matrix and the corresponding output pattern along the top. Any weights excited by both of the patterns are set to 1, whilst the others are left unchanged. The top of Figure 4 shows how two associations are stored in a CMM.

Searching the AURA memory: Recall

The data can then be recalled by performing a matrix multiplication between an input pattern and the CMM. If the input pattern is a binary vector, this calculates the number of active weights in each column that have been excited by the input, but vectors of integer values scores can also be recalled. The resultant (output) vector is then thresholded to find the output code associated with the stored pattern that most closely resembles the input pattern. There are many threshold methods that can be used, but the methods used in AURA Alert are:

1) **Willshaw** - A fixed threshold value is chosen and all sums greater than or equal to it are set to 1. All others are set to 0. This is used to locate the outputs associated with any stored patterns that have a given degree of similarity to the current input.
2) **L-Max** - The L highest sums are set to 1 and all others to 0. This is used to find the outputs that correspond with the closest matching stored rules.

The bottom half of Figure 4 shows the recall of two different patterns, using an L-Max threshold to find the closest match. The first input is a repeat of one of the stored patterns and the corresponding output is returned. However, the second input has not been seen previously, so the output associated the most similar input is returned. Each binary state is stored in a separate column of the CMM and repeated states are stored only once.

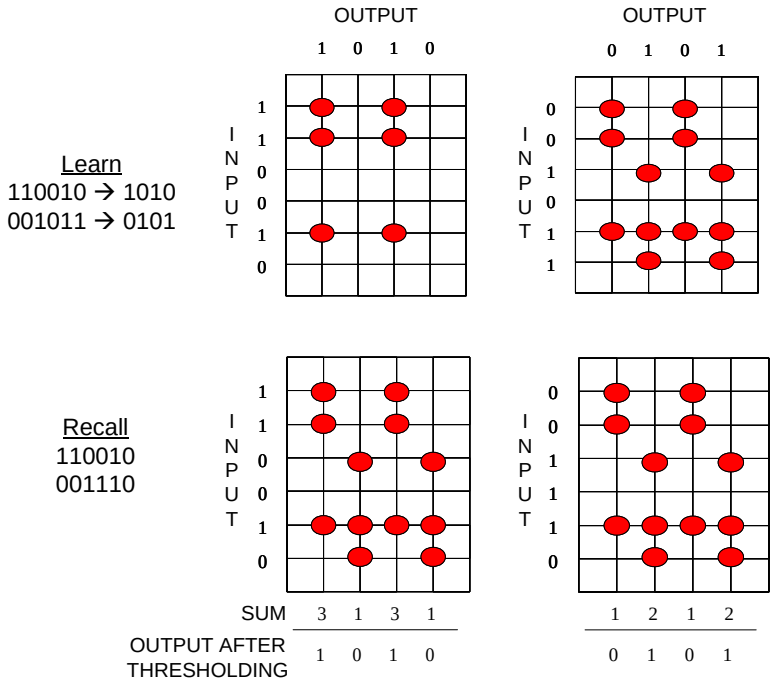


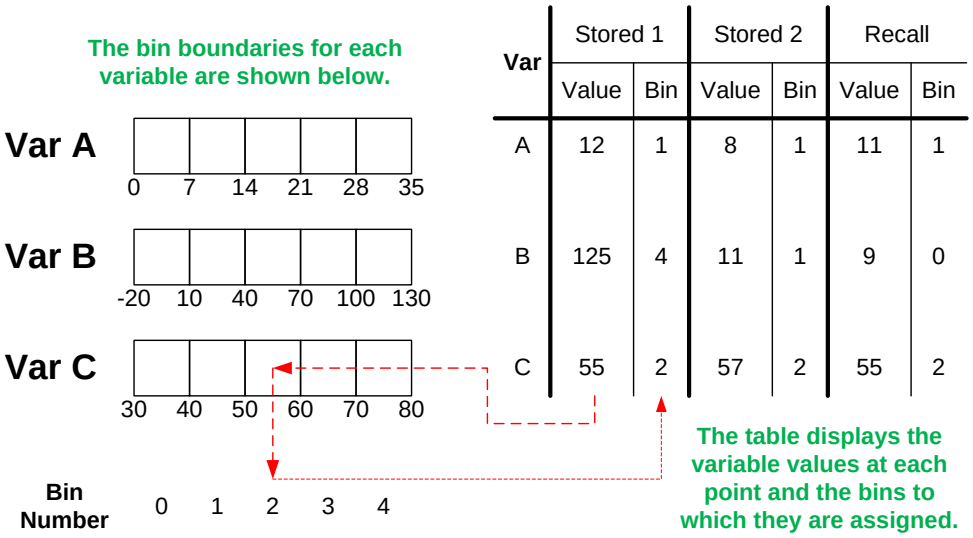
Figure 4 : Learning and Recalling Patterns using a Binary CMM

Once a representative period of normal behaviour has been stored in the AURA memory, the memory can be used to test whether the current state has been previously seen. The current state may be converted into a binary pattern via the same method used in the training phase. If the binary pattern is presented to the memory and a Willshaw threshold of N applied, where N is the weight of the input pattern, then the output pattern will have a weight of one only if the current state has been seen previously during normal behaviour. However, if the output pattern has a weight of zero, the current state has not been previously seen and the system will have to locate the most similar stored states so that the extent of the novelty can be evaluated.

An AURA memory can perform very efficient partial matching via a reduction in the Willshaw threshold. In this instance, by dropping the Willshaw threshold by X, it is possible to relax the matching process in the memory and search for any occurrences of the pattern where all but X of the variables belong to the same bin. However, this measure of similarity takes no account of the extent of the difference of the X variables that have not been assigned to the same bin. Consequently the system can fail to correctly identify the stored data points that most closely resemble the recall data.

An example of this problem can be seen in
Figure 5 where a recalled data point, containing three variables (A, B and C), is compared to two stored data points. The three values of the recalled state are much more similar to stored pattern 2 than stored pattern 1, largely due to variable B, which has a value of 9 for the recalled point and 11 for stored point 2, but 125 for stored point 1. However, when the values are assigned to bins, using the fixed width bin setup displayed on the left of
Figure 5, the value 9 is assigned to bin 0, 11 to bin 1 and 125 to bin 4. Therefore, the recalled data point matches two out of the three bins of both stored data points (Variables A and C) and even though the recalled pattern is much closer to stored pattern 2 than stored pattern 1, both are recalled by the same Willshaw threshold. Consequently, the recall system needs to factor in not only the number of bins that

match exactly, but also the distance between the assigned bins when they differ. This can be done by weighting the inputs.



Var	Stored 1		Stored 2		Recall	
	Value	Bin	Value	Bin	Value	Bin
A	12	1	8	1	11	1
B	125	4	11	1	9	0
C	55	2	57	2	55	2

Figure 5 : An example of the need for kernels during state recall. The distance between the recall vector and Stored 2 is much less than the distance between the recall vector and stored 1, but the number of matching bins is identical.

Weighting the inputs: Kernels

To generate a more accurate representation of the distance between the current and recorded asset states, the AURA scores can be used to apply weights to the bins, according to their distance from the current values. Rather than generating a binary pattern, a vector of scores is created, which defines a set of kernels that quantify the distance of each bin from the value. For example, if there were 10 bins for a variable and the value was assigned to the third bin, the score vector below could be used, which would generate a triangular shaped kernel around the recall value.

3, 4, 5, 4, 3, 2, 1, 0, 0, 0

If this scoring system were applied to the example above, the recall vector would return a score of 11 for stored state 1, whereas stored state 2 would return a score of 14. Consequently, the scoring system now more accurately reflects the actual distance between each stored point and the recalled vector. A detailed discussion of the various kernels that can be used to provide different approximations of distance can be found in [6]. However, the kernel defined by the equation below, which approximates to the Euclidean distance between the two points (if fixed width binning has been used), has been shown to provide the best results in testing so far. The equation below gives the value for bin number k ($bins_{fk}$), where the value of that variable has been assigned to bin t ($bins_{ft}$), max(n) is the maximum number of bins for any variable and n_f^2 is the number of bins for this variable.

$$Parabolic_{bins_{fk}} = \left[\left(\frac{\max(n)}{2}\right)^2 - \left(bins_{ft} - bins_{fk}\right)^2 \frac{\max(n)^2}{n_f^2}\right]$$

If this kernel were applied to a set of 10 bins, where the maximum number of bins for any variable was 10 and the value was assigned to the third bin, the following score vector would be used:

21, 24, 25, 24, 21, 16, 9, 0, 0, 0

The score vector, calculated from the chosen kernels, is presented to the AURA memory and a new threshold is required to generate the output pattern. To return only an exact match, a Willshaw threshold of N*S can be used, where N is the number of variables and S is the maximum score for one variable. This can then be slightly decreased to permit matches where some of the values are not assigned to the correct bin, but are assigned to one nearby. A threshold (T) between 0 and 1 is calculated during an evaluation stage and a state is classed as a novelty if there are no states returned when a Willshaw threshold of T*N*S is applied. An alternative strategy is to use an L-max threshold of 1, to find the stored state that best matches the current machine state.

If a link is provided between each column number and the raw data, AURA k-NN can be used as a filter to reduce a large number of stored states to a more manageable quantity of closer matches [4]. The L-max threshold is set to X and the system will locate the nearest X matching patterns.

The score associated with the most closely matching column(s) can then be used to determine how different the current state is to any that have been seen before, to provide a measure of the seriousness of the potential problem with the asset. Alternatively, the best match or matches may be passed to a back checking stage.

Back checking
By adding a system of back checks, where the original data from the closest matching states are compared to the query state, the system is able to more accurately calculate the distance between the points and provide a more informative output to the user, relating to the nature and extent of the differences.

5. Real World Examples: Energy Monitoring and Traffic Monitoring

Complex Asset Monitoring
The AURA Alert techniques have been used for the detection of irregularities in highly complex assets in a variety of different industries. In one example, AURA Alert is able to flag up novelties in the data several days before a catastrophic failure in a complex asset, without any prior knowledge of similar failures, even though no single variable exhibits any unusual behaviour or leave its normal operational range.

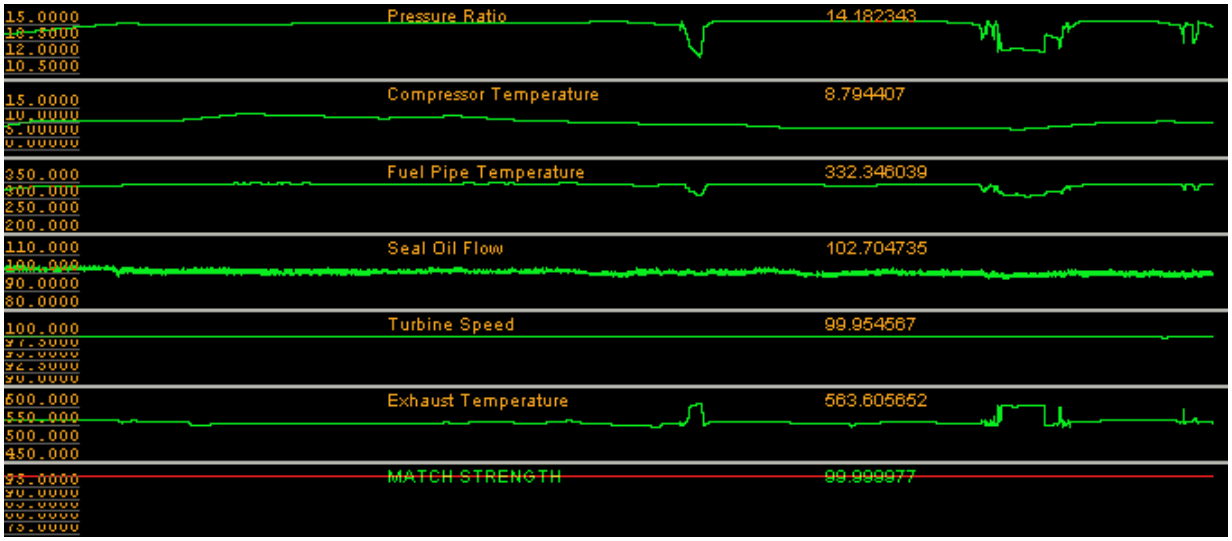
Of the several hundred variables available, twenty were selected that provided a good overview of the key components of the system. Operational data, sampled at one minute intervals, from a period of three months when the asset was considered to be working correctly, was presented to the AURA Alert system and the 1596 unique configurations encountered were stored in the AURA memory. An additional month of correct data was then used as a validation set to tune the threshold level, defining how different a recalled state must be to any stored in the memory to be classified as a novelty. The trained memory could then be used to search for novelties in historical data or to periodically check the current state of an online system.

The top six channels in each part of Figure 6 show the raw data from a subset of the selected variables for two periods of asset activity – section (a) shows a period where everything was running correctly, whilst section (b) shows the period immediately preceding a failure. When considering the variables independently there appears to be

nothing untoward in the data preceding the failure, as no parameters depart from their usual activity range or show unusual patterns. Consequently, the problems remain undetected until it is too late for preventative intervention.

However, by combining the variables together using AURA Alert, a clear warning is issued that the asset had deviated from its normal locations in the state space. The AURA Alert output can be seen in the "MATCH STRENGTH" channel, which has a value of 100 when the asset is in a previously seen state and drops down when a novelty is detected. The matching strength remains high during the period of normal activity in section (a), but in section (b) reports the presence of regular significant novelties, indicating that the asset state has departed from its usual operating behaviour. If the AURA Alert system was in place at the time, the engineers would have had sufficient time to investigate the cause of the alert and shutdown the asset before the major failure, even though that particular problem had never been encountered during the lifetime of the asset.

a) Asset data during normal running



b) Asset data prior to failure

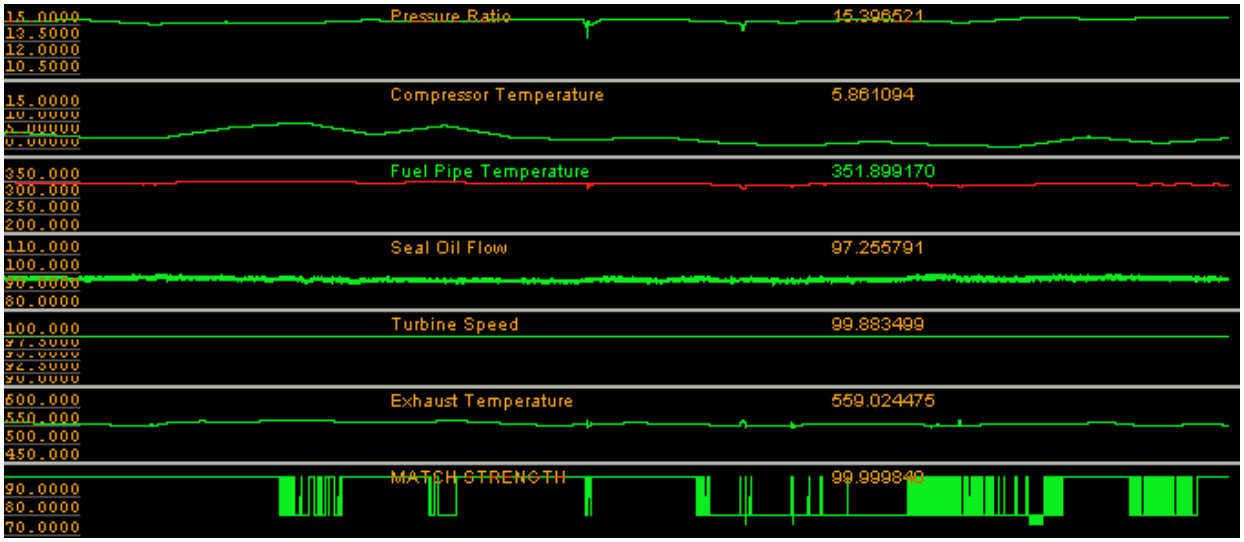


Figure 6 : The raw data from six of the variables and the AURA Alert matching strength for two periods of time. Part (a) is a period when the asset was running correctly, whilst part (b) shows the data immediately preceding a catastrophic failure.

In addition to reporting the matching strength of the state of the system at each time instance, AURA Alert is able to indicate which channels are the likely causes of the irregularities. By using an L-Max threshold on the AURA output, the most similar stored pattern to the current asset state can be obtained. By comparing the current state to the most similar state, the causes of the differences can be calculated and reported along with the matching strength. Figure 7 shows the novelties reported in each of the six channels shown previously, during the same time period as Figure 6, part (b). The reported novelties indicate that the compressor temperature and seal oil flow regularly deviated from their expected values, given the rest of the asset state.

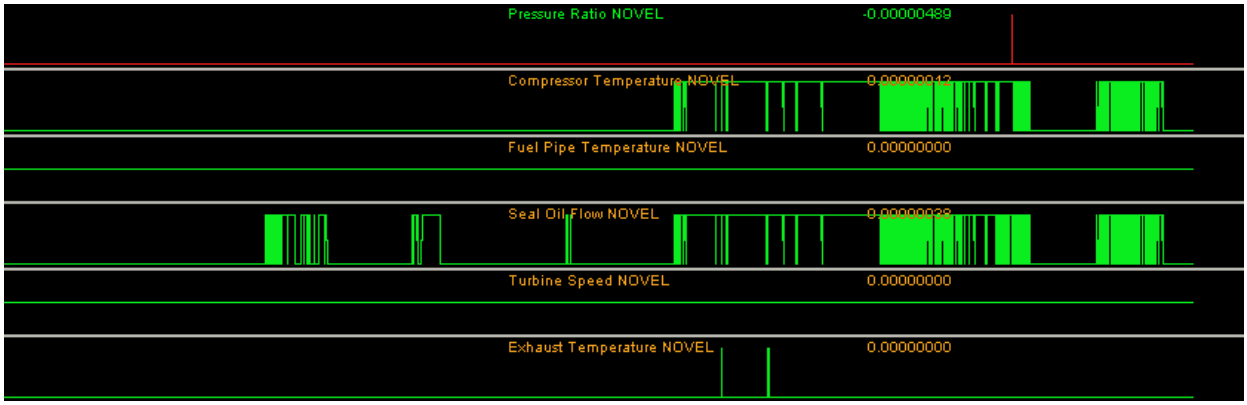


Figure 7 : The novelties that were reported in six of the channels, during the same time period as Figure 6, part (b).

Transport Applications – The FREEFLOW Project

The AURA pattern matching methods, described above, have been applied to real-time intelligent transport monitoring in the FREEFLOW project [7]. The objective is to identify traffic problems and to recommend traffic control interventions that will mitigate those problems. The AURA methods identify similar historical traffic patterns: time periods in the past when the traffic state, as represented by a set of traffic sensor readings, is as close to the current state as possible. The set of traffic control interventions implemented during these similar historical time periods are cross-referenced and recommended to the traffic operator for implementation. The AURA k-NN has been extended to distinguish between spatial differences in traffic patterns using a combination of the kernels described earlier and centre of mass techniques [8].

Tests of this system have been performed using data from Hyde Park Corner (HPC) in London, UK for 01/04/08 to 31/03/09. There are 32 traffic sensors in the HPC area. Sensor data is aggregated at 15-minute intervals and linked to traffic control interventions. The objective of the study is to determine if AURA can identify similar incidents in the historic data. Hence, five serious or severe congestion events in HPC area were identified to validate the pattern matcher: congestion in the northern/western arms of HPC, congestion in all arms of HPC, an equipment fault, a spillage and a broken down vehicle [8]. AURA matches historical time periods ranging from when congestion was developing until when congestion had dissipated. False positives (FP) indicate sensors that are congested during the historical time period but are not congested during the current time period. False negatives (FN) indicate sensors that are congested during the current time period but not during the historical time period.

Across the 5 incidents, retrieving the top 5 matches using 32 sensors, AURA k-NN with spatial matching produced 18 FP and 25 FN from a potential 800 sensor comparisons.

Note that this analysis is to some extent data dependent. There may not be any perfect matches in the historical data for novel situations so some FPs and FNs are inevitable.

The FREEFLOW pattern match methodology will be extended to a wider area of Hye Park Corner, to urban/inter-urban corridors around Maidstone, UK and inter-linked radial routes in York, UK.

6. Conclusions

The use of binary associative memories has been discussed for use in condition monitoring systems, in the (typical) situations where there is no *a priori* knowledge of emerging fault types or a full understanding of the complex underlying fault models. A particular instantiation of the use of associative memories has been discussed, called AURA-Alert, which has many advantages for application in this condition monitoring domain. The principle advantages being its ability to rapidly learn and model the normal operating envelope for a system, and the ability to search through complex high-dimensional multivariate spaces to detect deviations from normal conditions. These methods are scalable and robust, as well as being amenable to optimised hardware implementation, on architectures such as FPGA. The use of the methods for condition monitoring applications in transport and energy domains highlight the flexibility of the approach.

Acknowledgements
EPSRC/TSB/DfT Project Freeflow supported the work on traffic applications.

7. References

[1] Jim Austin. Distributed associative memories for high speed symbolic reasoning. International Journal on Fuzzy Sets and Systems, 82:223-233, 1995.

[2] Victoria J. Hodge & Jim Austin. A Binary Neural k-Nearest Neighbour Technique. Knowledge and Information Systems, 8(3): pp. 276–292, Springer-Verlag London Ltd, 2005

[3] V. J. Hodge & J. Austin. A Binary Neural k-Nearest Neighbour Technique. *Knowledge and Information Systems,* 8(3): pp. 276–292, Springer-Verlag London Ltd, 2005

[4] Willshaw, Longuet-Higgins, and Buneman. Non-holographic associative memory. Nature, 197: 960-962, 1966.

[5] Victoria J. Hodge & Jim Austin. Discretisation of Real-Valued Data in a Binary Neural k-Nearest Neighbor Algorithm

[6] Michael Weeks, Vicky Hodge, Simon O'Keefe, Jim Austin & Ken Lees. Improved AURA k-Nearest Neighbor Approach. In Proceedings of IWANN–2003, International Work-conference on Artificial and Natural Neural Networks, Mahon, Menorca, Balearic Islands, Spain. June 3-6, 2003. Lecture Notes in Computer Science (LNCS) 2687, Springer Verlag, Berlin.

[7] Glover, P., Rooke, A. & Graham, A. 2008. Flow diagram, Thinking Highways, 3(3)

[8] Krishnan, R., Hodge, V.J., Austin, J. & Polak J. A Computationally Efficient Method for Online Identification of Traffic Control Intervention Measures. Presented at, *42nd Annual UTSG Conference*, Centre for Sustainable Transport, University of Plymouth, UK: January 5-7, 2010