Pattern Matching in DAME using Grid Enabled AURA Technology

Robert Davis, Bojian Liang, Mark Jessop, Andy Pasley and Jim Austin
Advanced Computer Architectures Group (ACAG), Dept of Computer Science,
University of York, York, YO10 5DD.
[rob.davis, bojian, arp, mjessop, austin] @cs.york.ac.uk

Abstract

This paper describes research on the DAME Project into the use of Correlation Matrix Memories for imprecise pattern matching on large volumes of time series data using the Grid. This technology, named AURA has been developed over a period of 15 years at the University of York. In the DAME project, we have begun to apply AURA technology to the problem of searching aero-engine vibration data with the aim of finding partial matches to vibration anomalies and thus providing information that can be used to form an early prognosis / diagnosis of potential faults. Within the DAME project, the underlying AURA technology has been improved in terms of its speed and memory usage and adapted for use as a Grid service under Globus Toolkit 3. At the mid-point in the project (July 2003), research is ongoing into encoding techniques suited to fast processing of time series data. In the second half of the project, we intend to investigate the use of advanced data mining methods in combination with a distributed grid enabled implementation of AURA to support flexible and very high performance pattern matching over large amounts of data.

1. INTRODUCTION

The DAME Project

The DAME (Distributed Aircraft Maintenance Environment) Project is a pilot project supported under the UK e-Science research programme in Grid technologies and funded by the Engineering and Physical Science Research Council (EPSRC) Grant Number GR/R67668/01.

The DAME project involves four Universities: York, Oxford, Leeds and Sheffield and three commercial partners Rolls-Royce plc, Data Systems and Solutions LLC and Cybula Ltd.

The DAME project is described in some detail in another paper in this conference: "*Distributed Diagnostics on the GRID: DAME", by Tom Jackson et al*. The interested reader is directed to that paper for further information about DAME. In this paper, we focus on one of the key challenges presented by the DAME system: time series pattern matching.

Pattern Matching

Modern aero-engines operate in highly demanding operational environments and do so with extremely high reliability. They are fitted with extensive sensing and monitoring systems, the aim of which is to enable detection of the earliest possible signs of deviation from normal operating behaviour. In this respect, Rolls-Royce has collaborated with Oxford University to develop an advanced on-wing monitoring system called QUOTE.

QUOTE performs analysis on data derived from continuous monitoring of broadband engine vibration. The analysis is achieved through data fusion of the vibration data with instantaneous performance measurements. QUOTE is able to offer prognosis via the identification of *anomalies* in the vibration and performance data known to be precursors for certain types of fault – for example Foreign Object Damage. QUOTE is also able to identify *novelties* in the data that indicate behaviour that departs from normal, but cannot be identified as a precursor to any known fault.

Where QUOTE continuously monitors data for a single engine, it is envisaged that the ground based DAME system will compare novel patterns from engine vibration and performance data to data stored for previous flights. By identifying similar patterns in the data and analysing maintenance information corresponding to those flights, the DAME system will provide Maintenance Analysts and Domain Experts with the information necessary to correlate root causes and advise of appropriate remedial actions.

As each engine produces of the order of 1 GByte of vibration data per flight, the DAME *Pattern Matching* problem is a very challenging one. Consider that in a naïve implementation of such a search, the time series *query* data representing novel behaviour (100-1000 points) may need to be compared against every possible time offset

(100,000 – 1,000,0000s) in the time series vibration data for every flight (1,000s) of every engine (10,000s). This amounts to a search requiring potentially 10^{15} operations on Terabytes of data.

Despite the dramatic size of the problem, there is a need to return matching results in a time frame that is acceptable to the maintenance operation.

To address the pattern-matching problem the DAME project is exploring the use of AURA (Advanced Uncertain Reasoning Architecture) technology developed at the University of York [1] and commercialised by Cybula Ltd [5].

The next section of this paper gives an outline of the basic AURA technology and Correlation Matrix memories. Section 3 describes improvements to the performance of the AURA search engine developed under the DAME project. Section 4 outlines the Grid Service implementation of AURA-G under Globus Toolkit 3. Section 5 describes in detail the particular pattern matching problem posed in DAME, whilst section 6 outlines the encoding techniques used for time series pattern matching using AURA. Finally, section 7 outlines some research topics for the second half of the DAME project.

2. OVERVIEW OF AURA

AURA is designed to provide high performance pattern matching capabilities on uncertain and imprecise data. It achieves this by storing and searching binary patterns using a simple and analysable form of Neural Network, known as a Correlation Matrix Memory (CMM). This technique has been shown to be scalable to relatively large problems for example, text matching [2] and molecular databases [6].

Pattern matching using AURA requires two phases of operation, a storage or training phase and a search or recall phase. In the storage phase existing data, typically records from a file or database are encoded into binary sequences and stored in a Correlation Matrix Memory. In the search phase, new data is encoded in the same way and applied as an input to the CMM. The output of the CMM is then thresholded, decoded and the records identified fed into a *back-check* process.

Within an AURA based system, CMMs work as a high performance method of filtering out the vast majority of records that do not match the query. The matching records and a few *false positives* are retained and fed into the back-check process. The back-check performs conventional pattern matching operations to remove the false positives leaving just the records required.

We now give a simple example of how data is stored and searched in a CMM. The example is based on the idea of using a CMM to find anagrams of a given query - a word or sequence of letters in this case.

Encoding

Each item of data needs to be encoded as a binary *input pattern* and a binary *output pattern*. Finding an effective encoding scheme is both problem-specific and key to obtaining high performance. Developing a suitable encoding strategy is a fundamental step in implementing any AURA based system. Typical encoding strategies form simple binary patterns for each attribute of a record. These codes are then combined using either superposition (logical OR) or concatenation to form a binary pattern for the complete record.

Returning to our example, to form an input pattern for each word, first we use a 26-bit long binary code for each letter. So ‘A’ is encoded as ‘100000…’, ‘B’ as ‘0100000….’ ‘C’ as ‘0010000…’ and so on. These codes are then superimposed to form the input pattern for a given word. For example, ‘ACE’ is encoded as ‘101010000…’.

As well as an input pattern, an output pattern is also required for each item of data stored. The output pattern is used during storage and also during search, where the output of the CMM is decoded into a set of output patterns that identify the matching data items.

In typical applications a simple orthogonal encoding is used for output patterns. These codes have a single bit set, effectively associating a single column in the CMM with a single record. Such output patterns have the advantage of avoiding ghosting effects in the CMM as well as being trivial to decode.

Returning again to our example, we can use an output pattern for each word that has a bit set in a position reflecting the position of the word in the dictionary. So the first word has an output pattern of ‘100000….’, the second ‘0100000…’ and so on.

Storage

Storing data in a CMM is achieved by setting bits in the matrix corresponding to the outer product of the input and output patterns, or using simple

terms by setting bits in the cells given by the intersection of the rows selected by the set bits in the input pattern and the columns selected by the set bits in the output pattern.

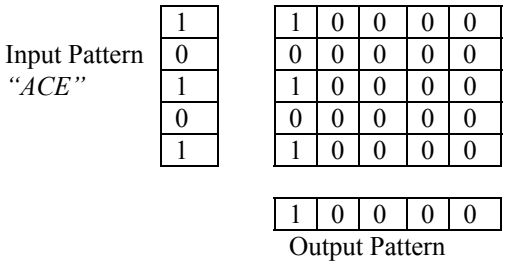

Figure 1

Figure 1 shows the word 'ACE' being stored in a CMM. Figure 2 below shows the state of the CMM after the words 'BAD', 'CAB', 'CAD' and 'DAB' have also been stored.

Search

To use a CMM to return matching data, we must first encode the query data using the same scheme that was used during storage. The input pattern for the query is applied to the input of the CMM, which evaluates the number of set bits that each column has in common with the input pattern. These column totals are then thresholded to generate a binary output pattern that can be decoded to obtain the matching records.

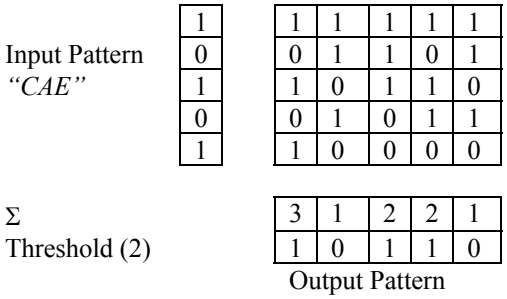

Figure 2

Returning to our example, let us assume the query data is 'CAE'.Figure 2 shows the CMM being evaluated for an input pattern of '10101000…' The column totals are 3,1,2,2 and 1 for columns 0 to 4 respectively.

To complete the search, the column totals are thresholded to produce an output pattern. In this case, we choose a threshold of 2 as we are interested in 'partial' as well as exact anagrams. This gives an output pattern of '10110…' Given the simple orthogonal encoding scheme used to generate output patterns, this can be trivially decoded to the 1st, 3rd and 4th words in the dictionary ('ACE', 'CAB' and 'CAD') all of which contain at least 2 of the letters in 'CAE'.

Interesting Properties of CMMs

Although the encoding scheme used in our example is very simple and not without drawbacks – for example how is 'AAA' encoded? It is hoped that it gives the reader a feel for how the basic technology works. It also helps to illustrate two key properties of CMMs.

Firstly, using CMMs it is possible to perform partial matches very efficiently. Suppose we had a CMM that encoded 20 different properties of a set of records in a database. A single CMM search could be used to identify all the records that matched 10 or more of the attributes of the query data. By comparison the number of separate queries required using standard database techniques would be over 184,000 (all combinations of 10 out of 20).

Secondly, the performance of a CMM based search can far exceed that of an equivalent sequential scan through the data. Take the simple anagram example. Suppose we were looking for three letter anagrams, a simple sequential scan of all the three letter words against a query might reasonably take *O(3N)* operations were *N* is the number of three letter words in the dictionary. Using a CMM, we note that only 3 out of 26 rows are activated by the input pattern. (The input pattern is said to have a *saturation* of 3/26 = 0.12) The total amount of information that needs to be processed is reduced by a factor of *1/saturation* – assuming that the data is spread reasonably evenly between the rows and the CMM is implemented effectively. For more complex applications, such as the Address Matcher [2] using a smart encoding scheme can bring the performance improvement resulting from very low saturation input patterns to 100 to 1000 fold.

3. AURA PERFORMANCE

To achieve the best possible performance, it is necessary that the internal data representations used in the CMM be implemented in a highly efficient way. With this in mind, during 2002/03 the AURA software library was given a 'from the ground up' redesign starting with the specification of a new AURA Application Programming Interface (API) developed in conjunction with the experienced AURA user base.

The aim of the redesign was three fold:

1. To improve performance of the AURA library in terms of both memory usage and search times.
2. To make the library easier to use.
3. To engineer the library to commercial software standards.

As a result, the AURA library now has a set of comprehensive user documentation, including a User Guide and Programmers Reference Manual.

The new AURA library incorporates new features designed to enhance performance. These include the use of a variety of internal representations for the binary data. An additional optimisation phase, between storage and search chooses the best form of compression, for performance or memory, given the data actually stored. In addition, the internal organization of data is designed to be cache-friendly and thus retain high performance as the amount of data stored is increased.

Benchmark performance tests have been carried out on the new library against the previous implementation of AURA. These are reported in detail in a technical report [3].

Figure 3 shows the memory requirements in Mbytes of the new AURA library compared to the old library for 500,000 records of address data.

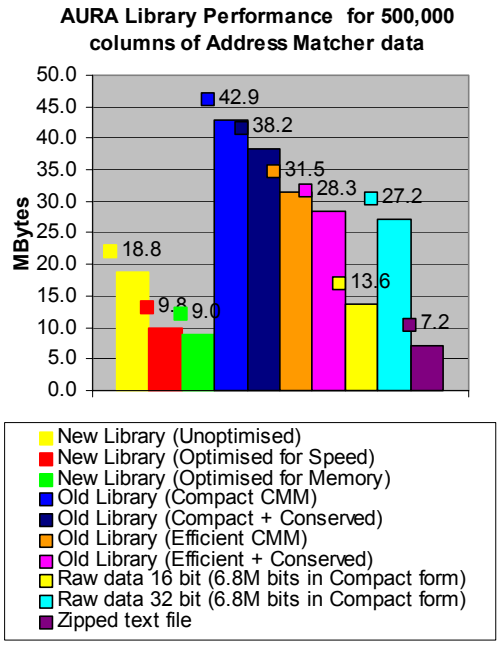

Figure 3

The columns on the graph show the memory usage for different optimisation strategies using the old and new libraries. Overall, the new library shows a significant improvement in the memory required, typically needing only 35% of the memory required by the old library (9.8 Mbytes v. 28.3Mbytes): Approximately a factor of three improvement. As an indicator of the efficiency of the compression achieved by the new library, the memory required to store the raw data in zipped form is 7.2 Mbytes.

Figure 4 shows the average search times in milliseconds to search a CMM of 500,000 addresses For typical searches of the postal address data, the new AURA library optimised for speed provides search performance approximately three times faster than the old library (3.0 v. 9.5ms). This improved performance is thought to be due to the internal structures used in the new library making evaluation of search results far more 'cache friendly'.

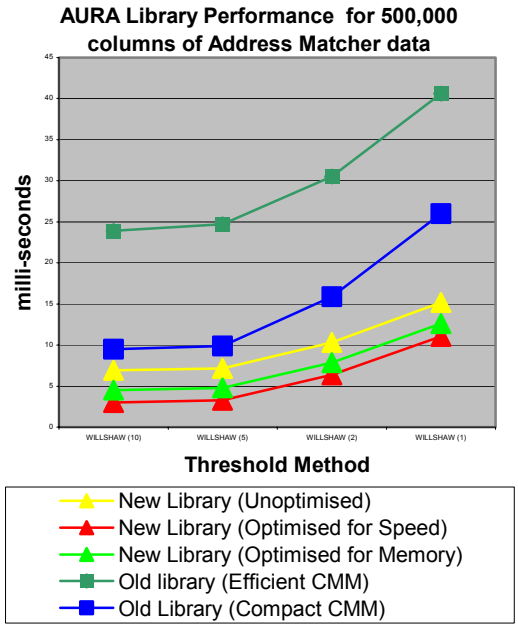

Figure 4

Interestingly, optimising for minimum memory also results in improvements in search time over the un-optimised data. This is because the postal address data has a reasonable amount of coherence between columns. For this data, there are internal representations that are effective both as a means of reducing the memory required and the execution time of the search.

4. AURA-G

The new AURA library has been implemented in the grid paradigm using Globus Toolkit 3 OGSI grid services [7]. Currently the library along with any application specific code forms a two-tier architecture.

Tier one hosts a generic AURA service with the second tier containing application specific code. Clients interact directly with the second tier, allowing application developers to abstract away from the pattern match domain.

The generic AURA service is implemented as a transient service, using Java for the grid and search service APIs. Where performance is key, the AURA library is embedded in the service as a native code. The single user library has effectively been converted to a multi-user service that is capable of handling many users working with single or many CMMs.

A static DAME registry has been built that contains the handles of AURA service factories along with any instantiations. The second tier is then able to create as many search engines as required. With this mechanism, a distributed search service can be provided.

Currently only one application specific service exists in the second tier. This service is responsible for training and searching using the AURA service. Data is provided to this service in an application specific way, and any results are returned from the service in the same format. Internally this service knows how to convert data to the correct format to build a search, and how to interpret the search output. At this time, the DAME application uses a single instance of the second tier service.

In the future, the AURA architecture will move to three tiers. This will move the AURA native code out of the AURA service and onto targeted hardware consisting of grid machines, Linux clusters and potentially redundant office PCs. The next iteration of the AURA library should make it possible to distribute single or many CMMs across multiple machines of heterogeneous platforms. Splitting the architecture in this way, to split the interface from the processing will boost the overall AURA performance and provide a more responsive service.

5. PATTERN MATCHING IN DAME

Figure 5 shows a Z-mod plot, used to illustrate the raw broadband vibration data. The x-axis represents time, whilst the y-axis represents frequency. The intensity of the colour represents the amplitude of vibration at that time and frequency.

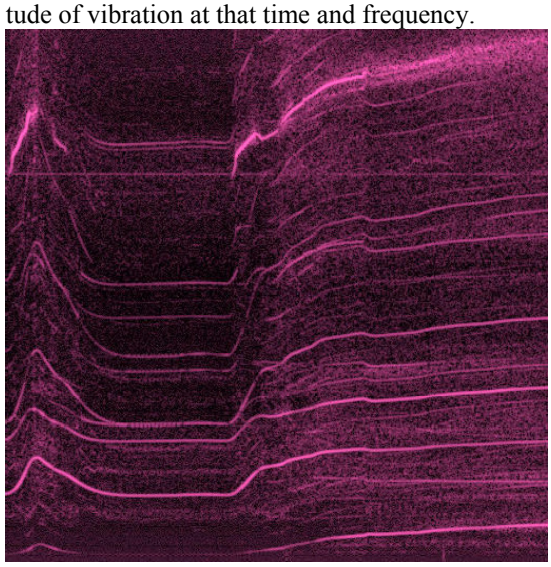

Figure 5

The lines on the Z-mod plot represent the peaks of different harmonics related to the speed of various components within the engine. Each of these harmonics can be extracted as a separate set of time series data, referred to as a tracked order.

Novel patterns in the tracked orders can indicate a pre-cursor to a potential fault. A key pattern-matching problem for the DAME system is therefore to identify similar patterns to any novelties detected by QUOTE or flagged by a Domain Expert. Identifying similar patterns in the data is a key step on the way to correlating the root cause of the problem.

Time Series Subsequence Matching

The pattern-matching problem in DAME is an example of what is referred to as a '*query by example*' or '*query by content*' time series subsequence-matching problem.

Assume we have a novel pattern of n data points – the query. Further let us assume that the database of existing time series contains m points (typically m is very large). In a simple approach referred to as *sequential scan*, we must compare the n points in the query against the sequences in the time series database for every possible time offset. This leads to a computational complexity of $O(nm)$. For

large databases, this can be infeasible. Methods are therefore required that reduce the amount of computation required to find close matches. This is a particularly active area of research.

On the DAME project, we have taken the approach of first trying to solve the pattern-matching problem in a simple way using conventional techniques and a sequential scan. Further research effort will then be deployed investigating ways in which the performance gains possible using CMMs can be brought to bear to solve the subsequence-matching problem in a highly efficient way.

In the next section, we describe the approach used to perform time series subsequence matching using conventional techniques.

Delta Modulation (DM) codes

There are several standard signal-encoding techniques appropriate for vibration time series data. These are PCM – Pulse Code Modulation, DPCM – Differential Pulse Code Modulation and DM – Delta Modulation. PCM directly encodes the value of the signal while DPCM and DM encode the differential w.r.t. time.

DM is a widely used technique in communication. It is a special case of DPCM where there are only two quantisation levels. A start value v_0 is set before encoding, a predictor, usually a feedback Digital to Analogue Convertor generates an estimation signal z_i and the difference between the current input signal x_i and z_i, $e_i = x_i - z_i$, is encoded. After a short start interval, the differential code will trace the input signal with an error that is less than the quantisation step. The differential code c_i is given by:

$$c_i = ADC(x_i - z_i),$$

Where $ADC()$ represents the action of the Analogue to Digital Conversion, z_i is the estimation of input signal x_i. Let Δ be the quantisation step, z_i is given by

$$z_i = v_0 + \Delta \sum_{j=0}^{i-1} c_j$$

The code c_i only needs two states, one for positive e_i and one for negative e_i.

To avoid overload, DM requires over-sampling to ensure that sharp changes in the value of the signal can be tracked faithfully. Over-sampling results in the use of interpolation points and an increase in the size of the binary code.
To obtain better performance on the vibration data, the DM code was extended to 3 states. Compared to a two state code, the three state DM code can follow the original signal in a more precise way without the need to go through a low-pass filter. A two state DM code tracks a steady state signal as alternate positive and negative codes. For a three state DM code, the maximum quantisation error is half the quantisation step.

The time series representing tracked orders in the vibration data were empirically found to require 7 interpolation points using 3 state (2-bit) DM codes, with a further code added to adjust for drift. Hence each point in the time series data is represented by 8 2-bit DM codes, 16-bits in all.

To detect similarity between two sub-sequences, a correlation measure was used based upon the co-variance of the sequences. (Co-variance is a measure of the tendency of the sequences to vary in the same direction). This measure differs from the standard *Euclidean Distance* similarity measure in that it finds patterns to be similar in the presence of differences in amplitude.

The correlation between the query and a given offset in a stored time series is found using the following pseudo-code:

```
For each offset in the time series {
   For each byte in the query {
      Form a 16-bit index from the byte
      of the query and the correspond-
      ing byte of the time series
      Look up the correlation value in
      a 64K table of correlation values
      Increment the similarity score
      for this offset based on the
      value found
   }
}
```

Where the number of data points in the stored time series is *m* and the number of data points in the query is *n*, this method requires effectively *O(2nm)* operations.

Figures 6 and 7 show a query pattern and a candidate match found via 3 state DM coding and a sequential scan of the data.

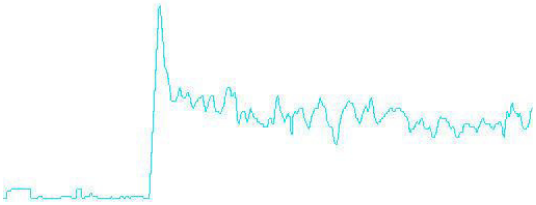

Figure 6

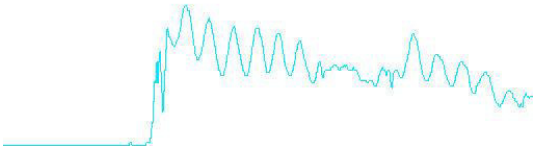

Figure 7

We note that using a simple DPCM coding supporting the –8 to +8 values that the 8 DM codes for each data point can take would require 8-bits per data point rather than 16, reducing the overall computation to *O(nm)* operations

The efficiency of the simple sequential scan can be further improved as noted in [4] by short cutting computation of the distance measure for each offset once the value reaches that of the smallest distance seen so far, or in the case that the K-nearest patterns are required, the K-th best value seen so far. This can reasonably be expected to improve the performance of the sequential scan by a factor of 3 to 5[4].

6. AURA TIME SERIES MATCHING

Following on from the work on DM and DPCM codes, a very simple time series pattern matching application has been developed using AURA-G.

The encoding scheme employed takes the DPCM value (-8 to +8) for each data point and transforms this into a single set bit in a binary code of 17-bits, where '10000…' corresponds to –8, '010000…' to –7 and so on.

To form a binary pattern for sequences of data points, the 17-bit codes are concatenated. Thus for a subsequence of 100 data points, we have a binary code of 1700 bits which contains 100 set bits.

Initially, a simple scheme was used to create the CMM, with each column representing one of the *m* possible offsets within the time series, and holding values for the *n* data point sub-sequence starting at that offset. Although this approach provides a simple mapping of the problem, it is infeasible in practice due to the large memory usage *O(nm)* a factor of *n* more than the original time series.

This problem was overcome via a data folding technique. Here, the data for each stored time series is divided into contiguous segments of length *n*. These segments are encoded and stored as the columns of a CMM. Thus there are now only *m/n* columns by *n* data points, which have an equivalent memory requirement to the original time series. The data folded CMM does however require *n* searches to generate scores for all possible offsets and thus complete the matching operations. Provided that each row in the CMM still contains a reasonable number of set bits, then each search is proportionately faster, making the data folded CMM as fast as the naïve implementation whilst requiring a fraction of the memory.

Distance Measures

Using the simple encoding scheme described above, it was not simply sufficient to apply the code for a query to the CMM and see which offsets are returned as potential matches. This would be a very odd similarity measure returning subsequences that have a small number of precisely matching data points but vary widely in their overall shape.

Instead a CMM based implementation of a suitable similarity measure such as Euclidean Distance or co-variance is required. Such measures can be calculated by applying a weighted *kernel* to the CMM.

For example, assume that we have a possible range of values 1-5 and wish to calculate a Euclidean Distance measure. The maximum distance is 4, whilst the minimum distance is zero. As CMM thresholding returns higher values as matches, we can use kernel weightings of $16 - d^2$, where d is the distance between each possible value (1-5) and the data for the query (4).

Figure 8 illustrates the kernel used for rows 0 to 4 (representing integer values 1-5) and a query value of 4. The first column in the diagram (all 1s) indicates that all the rows in the CMM are activated. The second column gives the row weights used in the kernel.
Given weights on the input pattern, the CMM evaluates the active rows (0-4) by adding the row weight to the column total whenever there is a set bit in the column.

	1	7	1	0	0	0	0
Input Pattern	1	12	0	1	0	0	0
	1	15	0	0	1	0	0
Data = 4	1	16	0	0	0	1	0
	1	15	0	0	0	0	1
Σ			7	12	15	16	15
Thresh-old (15)			0	0	1	1	1

Output Pattern

Figure 8

To obtain matches within a given distance a suitable threshold has to be used. In the case of our simple example, a threshold of 15 returns all matches within a distance of 1.

Using a distance measure based on co-variance and suitable kernels it has been possible to build a simple demonstrator showing that time series sub-sequence matching can be achieved using AURA technology.

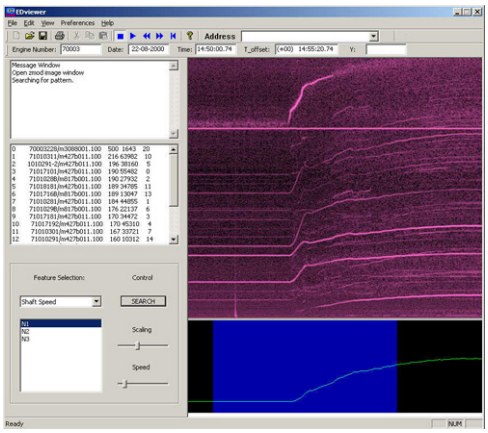

Figure 9

Figure 9 shows a screen shot of the graphical tool used to demonstrate this pattern-matching capability in DAME. The screen shot shows the query pattern, highlighted in blue. The search returns a set of candidate matches ordered by their degree of similarity to the query. These can be individually selected and displayed.

7. RESEARCH DIRECTION

Our initial work on time series pattern matching provided a simple proof of concept for the DAME mid-term demonstrator. It has also served as a catalyst highlighting the need for further research into high performance time series pattern matching using AURA.

Key areas to investigate are:

1. Similarity measures and their relevance to the pattern-matching problem in DAME. For example, similarity measure that enable patterns to be matched in the presence of noise, outliers and scaling in amplitude and time.
2. Combining dimensionality reduction techniques with CMM based evaluation for high performance indexing of time series data.
3. The approximation of similarity measures using smart encoding techniques and kernels

REFERENCES

[1] AURA website
http://www.cs.york.ac.uk/arch/nn/aura.html

[2] *"Improving automated postal address recognition using neural networks"* DPhil thesis, David Lomas, University of York 2001 BLDSC NO. DXN042056

[3] "*AURA Library Performance Analysis"* AURA/YORK/TR/02.010. Robert Davis 7th March 2003

[4] Eamonn Keogh, Shuruti Kasetty. *On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical demonstration.* SIGKDD 2002, Edmonton, Alberta, Canada.

[5] Cybula Ltd website www.cybula.com

[6] J Austin, A Turner, K Lees, *Chemical Structure Matching using Correlation Matrix Memories* Proc. ICANN 99, Endinburgh, September 1999, IEE Conference Publication 470, IEE, London.

[7] S Tuecke et al, *Open Grid Services Infrastructure (OGSI)*, The Global Grid Forum, http://www.gridforum.org/ogsi-wg/