The Automatic Explanation of Multivariate Time Series

Allan Tucker
Birkbeck College
University of London

Abstract

Due to the advances in data capture and storage techniques over the last decade, the size of Multivariate Time Series (MTS) data being recorded has grown massively. Many of these MTS are characterised by a large number of interdependent variables with large possible time lags. If new and useful knowledge is to be automatically learnt from this type of data in order to aid the understanding of the underlying processes, a paradigm must be identified that is capable of modelling data with these characteristics but at the same time exhibiting *transparency* in how it models the data. A key challenge is that the number of possible models is very large since it does not only depend on the number of time series variables, but also on the size of possible time lags between 'causes' and 'effects'.

In this thesis a general framework is described for automatically learning probabilistic models from MTS with large time lags and high dimensionality in order to explain the underlying processes involved. Specifically, a novel method to learn dynamic Bayesian networks for explanation from these series is developed. This involves an efficient pre-processing stage, which effectively groups MTS variables in order to reduce the dimensionality of the problem. After pre-processing, a combination of Evolutionary Programming, Genetic Algorithms and heuristics is used to speed up convergence when learning models. In addition, an approach is looked at for the off-line learning of dynamic Bayesian networks with changing dependency structures. All experiments have been carried out on a mixture of synthetic and real data taken from an oil refinery repository. The resultant models are used to generate explanations that are evaluated in several ways, including reviewing the feedback from chemical process engineers. These results have demonstrated that the proposed framework is very promising in terms of both efficiency and accuracy.

Acknowledgements

I would firstly like to thank my Supervisor Xiaohui Liu for the support and advice he has given me throughout the last few years. I would also like to thank Stephen Swift for his excellent advice and constructive criticism of my work. Thanks must also go to both of these people for their company as valued drinking partners during our in-depth discussions, both in our local pubs and on our various travels.

Without the oil refinery data and process knowledge, this thesis would never have come to be and so I must also thank Andrew Ogden-Swift and Andrew Trenchard from Honeywell Hi-Spec Solutions, UK as well as Donald Campbell-Brown and Lorcan Cleary from BP-Amoco for assisting me in obtaining the data and explaining the various processes involved in the chemical process. Thanks also to the various members of Honeywell Technology Centre, Minneapolis, USA, for allowing me to present my work to them, and learn from their experiences of problems similar to those that I encountered during my research.

I am grateful to everyone at Birkbeck College who has supplied me with the everyday requirements of a PhD student, demanding as these can be, including the systems group, the secretaries and George Loizou. Thankyou to everyone who has read my various thesis drafts without falling asleep and for giving invaluable feedback.

Lastly, a big thankyou to my friends and family for their support during all the good and bad times and, in particular, to Jen without whom I would not have been able to struggle through the challenges that my research and life in general have thrown at me.

"Time involves change. We say that something can remain unchanged through time but there could be no time if nothing changed."

J. M. E. McTaggart. (The Unreality of Time)

Contents

List of Figures

List of Tables

Table 6.2. Structural Difference Results using HCHC.

List of Algorithms

1 Introduction

1.1 The Problem

Due to the advances in data capture and storage techniques over the last decade, the size of repositories storing multivariate time series (MTS) data has grown massively. Many datasets now exist that are very high in dimensionality e.g. gene expression profiles, financial time series and industrial process data. This data will be characterised by a large number of interdependent variables, though some may have no substantial impact on any others. Many of these complex systems such as industrial and financial processes record data at frequent time periods and in some cases there can be large time lags between causes and effects. If we want to try and learn useful new knowledge about this type of data in order to aid the understanding of the underlying processes, we need to identify a paradigm that is capable of modelling data with these characteristics but at the same time exhibiting *transparency* in how it models the data.

There has been a great deal of research into the analysis of time series data in both the statistical and artificial intelligence communities, especially for *forecasting* purposes. In this context, there are a variety of statistical methods for modelling MTS, e.g. the vector autoregressive process, Markov Chain Monte-Carlo methods and other non-linear and Bayesian systems [Lutkepohl93, Pole94]. In the computing community, many forecasting methods have also been proposed using recurrent or time-delay neural networks, evolutionary computation, inductive logic programming and, more recently, support vector machines.

In modelling MTS for *explanation* purposes, work in the statistical community has been largely focussed on the use of variation in one or more series to explain the variation in another series e.g. multiple regression models or linear systems [Chatfield96]. When it comes to modelling MTS for explaining current observations on all the variables using previous observations of the same variables, the computing, especially AI, community has explored a number of approaches over the years. Probably the most common approach has been the use of relevant knowledge or expertise from domain experts to construct the explanation model [Shahar97]. This process requires am intensive knowledge acquisition effort and the resultant model is often incomplete and inconsistent with the observations. Therefore, a considerable effort is required to refine and fine-tune the model.

AI researchers have also proposed *model-based* approaches to construct explanations in which an underlying system based on first principles in a domain is used. These model-based systems relay on low-level mathematical principles and have been used to successfully model complex and uncertain data [Chang94, DeKleer91, Kramer87, Petti90]. They have been used to monitor, detect faults and isolate their causes. However, the construction of such models is extremely costly in terms of expertise and has, therefore, only been used in special circumstances. What is more, these model-based paradigms offer little in terms of explaining how events in complex systems arise - the model that is used often contains a complex set of mathematical equations that will not aid the understanding of the underlying mechanisms at work in these systems. Model-based approaches rely on well-understood domain theory and an efficient way of reasoning with the system description to generate explanations.

Between the 1960s and 1980s, expert systems were investigated in order to automatically diagnose identified events within many real-world processes, including medical, scientific and engineering applications, [Shortcliffe76, McDermott84, Dhurjati87]. Whilst these systems were capable of modelling some domains in a way that humans could understand (i.e. they were transparent), many were found to be inadequate in modelling complex processes due to the dynamics and uncertainties that occur within systems such as chemical processes. These *experience-based* approaches assume that there is a rich body of knowledge and experience from domain experts about the MTS under consideration, the cost of eliciting the knowledge from experts is reasonable and one is prepared to continuously resolve any inconsistency between the model which is largely derived from human experience, and the data collected from real-world operating environments. When these assumptions do not hold, an alternative must be found.

A general framework is required that will automatically learn models from MTS with large time lag and high dimensionality in order to explain the underlying processes involved. It must be able to deal with modelling complex processes but at the same time the input and output of each stage in the learning process must be transparent so that it is explainable unlike previous model-based systems. This thesis introduces a general framework for learning such models.

1.2 Explanation

Within this thesis we will often refer to the notion of *explanation*. We define an explanation to be a set of possible events leading up to some previously identified observation with the

aim of giving an insight into why that observation has been made, where an event is a variable or set of variables being in a particular instantiation. The set of variables that can be used as observations to be explained will be the same set of variables that are used to construct the explanation. It should be made clear that this definition of explanation does not include the notion of direct *causal* influence between events. This will be described in more detail in section 2.3. The model that we want to use for explaining datasets will contain *uncertain* information. Firstly in some systems, there may be uncertainty in the recording of data. For example, there may be errors in the instruments when measuring real world data that means that we may not be one hundred percent sure that it is correct. Secondly, having learnt a model for explanation, there will be uncertainty within the process of reasoning about events in the data. For example, if we were to observe that variable A is very high, and the reasoning process concludes that this is most probably due to variable X being low 5 minutes ago, we will not necessarily be one hundred percent sure that this conclusion is correct. The model that we employ must be able to manage these sorts of uncertainties in the data and reasoning process.

1.3 Methodology

In this thesis we describe a general framework for learning probabilistic models from MTS in order to explain the underlying processes involved. This will be beneficial to many applications in medicine, science and engineering, for example the control of chemical processes that we will focus on for the scope of this thesis. In reviewing process data such as that from an oil refinery, process engineers often come across trends with unexpected characteristics. In many cases these anomalous events have a significant adverse economic

impact, whether in terms of reduced yield, excessive equipment stress, or violation of environmental constraints. The identification of such events is important but of greater importance still are adequate explanations of how these events arose. These could then be used to modify operating practices, retrain operators or conduct anticipatory planning. The MTS that we propose to model are *high dimensional* datasets with *large time lags* and *changing dependencies*.

The specification for such a tool (based on [Samad97, Ogden96]) will be to:

a) Learn a model from new data as quickly as possible (in a matter of minutes) with as little user intervention as possible. The model must be transparent and, therefore, be capable of making the relationships found within the data explicit to non-statisticians or mathematicians.
b) Learn a number of models from large repositories of data that contain changing dependency structure. Once again all models must be transparent, including the way dependency changes are modelled.

(a) addresses the problem of trying to understand why a particular variable or set of variables have started behaving in some way when time is limited. For example, when data has been generated recently from a high dimensional process and a model must be learnt rapidly from this data in order to assist the understanding of the underlying processes and the explanation of certain variables.

(b) addresses the problem of trying to learn models from the vast repositories of historic data that are common in many scientific and engineering applications. The process is carried out off-line and, therefore, time is not as short, although efficiency is still required as the search spaces can be massive when taking into account the possible changing dependencies.

The resulting models will allow users to query events in the system. For example, users can ask for probable reasons that variable A is currently high whilst variable B and variable C have been low for the last 5 minutes. Output would be of the form: Given the evidence, the most likely explanation is variable X being low 3 minutes ago with probability 0.783 which is, in turn, explained by variable Y being low 10 minutes previous with probability 0.679.

Various heuristic searches are employed to find a good set of probabilistic models for each MTS. Searches include hill climb and evolutionary approaches and employ various heuristics which are compared and contrasted in Chapter 3. The framework involves breaking down high dimensional MTS into smaller dimensional MTS that are more suitable for learning models from in a feasible amount of time. This is achieved as timely as possible through the rapid search for high correlations within the data followed by *grouping* the variables based on these discovered correlations and a grouping metric. This subject is dealt with in detail in Chapter 4. Having broken down the single MTS into smaller dimensional MTS, a new method is introduced in Chapter 5. Here, the proposed method is systematically compared to the best performing one from Chapter 3 and its efficiency and accuracy analysed. The MTS we model contain dependencies that change over time and in Chapter 6 a method is investigated for incorporating the segmentation of MTS variables into different *operating*

states whilst searching for good probabilistic models simultaneously. This method should still continue to be transparent and allow explanations to be generated based on different operating states.

The basic framework adopted for solving the problem is illustrated in Figure 1.1. Data Preparation involves the selection of variables and portions of MTS from the oil refinery data as well as the generation of the synthetic data (using vector auto-regressive and Bayesian network models). Variable Groupings involves pre-processing the high-dimensional MTS into several smaller dimensional MTS and is addressed in Chapter 4. Search Methods involves looking at existing search procedures for Bayesian Networks and investigating their performance when adapted to learning DBNs (See Chapter 3). It also involves comparing a new proposed algorithm to the best of the existing search methods (Chapter 5). Changing Correlations addresses the issue of extending the models to handle relationships that change over time whilst keeping the generated explanations transparent (Chapter 6).

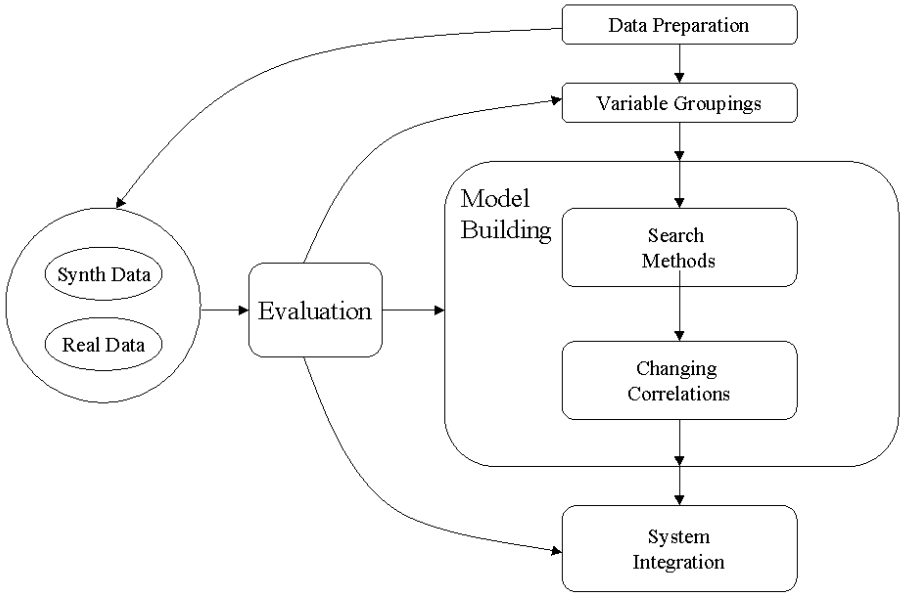


Figure 1.1. The General Methodology adopted within the Thesis for Learning MTS Models

1.4 Contribution to Knowledge

The five key contributions to knowledge that this thesis provides are outlined in this section.

1. Grouping - A new method is introduced and investigated for *grouping* MTS variables where **dimensionality** is very **high**, **time lag** can be **large** and/or **time** is **short**. This method consists of a correlation search using an EP combined with a grouping algorithm for pre-processing MTS. It has been shown to be very efficient (results within 4 minutes on MTS with 50 variables, 1000 time points and time lags of up to 60, using a standard Pentium) compared to the other methods investigated, including a clustering algorithm, hill climb and various approximate searches. What is more, it

can be scaled to very large MTS in order to reduce the dimensionality of model building.

2. Comparison of Searches - A comparison of standard static BN search methods that have been adapted to search for DBN structure including Suzuki's Branch and Bound [Suzuki96], Cooper's K2 [Cooper92], Larranaga's GA {Larranaga96] and Wong's EP [Wong99]. These have been tested on both information theory and statistical metrics.
3. EP-Seeded GA - A new method for efficiently searching for DBN structure known as EP-Seeded GA. This makes use of a rapid approximate correlation search using EP in order to seed the original population of a GA. The EP has been found to be excellent at rapidly homing in on structure in the MTS (within the time axis). By seeding a GA in this way, as well as defining an operator that will maximise recombination, results have been obtained that show the algorithm to be highly efficient (results within about 3 minutes on MTS with 20 variables, 1000 time points and time lags of up to 60, using a standard Pentium) compared to the other methods investigated.
4. Incorporating Changing Dependencies - A method for incorporating changing dependencies into the representation whilst ensuring transparency has been achieved with some success. This is achieved by using hidden discrete variables to represent the changing states of dependencies. These variables retain transparency in that they can easily be interpreted as `the operating state of the system'.
5. Application to Synthetic and Chemical Process Data - The above methods have allowed the rapid automatic explanation of synthetically generated data and chemical process data through using a combination of grouping and EP-Seeded GA so that incoming data can be analysed `on the fly'. Modelling changing dependencies allow

models to be built from vast historical data repositories and then queried with as little or as much user intervention as required. Transparency has been ensured throughout so that at the end of each process (e.g. grouping, model building etc.) a non-statistical user can intervene and adjust the models.

1.5 Road Map to this Thesis

This section contains a brief guide to the contents of each chapter within this thesis.

Chapter 2 looks through various dynamic models in the context of explaining MTS automatically in order to justify the use of the dynamic Bayesian network. These models include statistical, neural network and state space models.

Chapter 3 firstly outlines the associated issues of Bayesian networks, both static and dynamic, such as learning from data, inference, discretisation, changing dependencies and hidden nodes. The second part of this chapter takes some of the widely used standard methods for learning static Bayesian networks, adapts them to be able to learn dynamic models through the use of a new representation and compares the methods for efficiency and accuracy on several synthetic datasets. This chapter is based on the work in [Tucker2001b]

Chapter 4 addresses the problems associated with very high dimensionality in MTS, and introduces a new method for pre-processing data by breaking MTS into several lower dimensional MTS based on dependencies discovered in the data. The method allows the user to make a trade-off between speed and accuracy by altering certain parameters. It is applied

to synthetic data and the oil refinery dataset and the results are documented. This chapter is based on the work in [Tucker2000, Tucker2001a]

Chapter 5 introduces a new method for learning dynamic Bayesian networks by seeding a Genetic Algorithm with the results of an Evolutionary Program search for strong dependencies. This algorithm is compared with the efficiency and accuracy of the best method found in Chapter 4 when applied to synthetic data and the oil refinery data. Sample explanations are shown which have been generated from resulting networks for both the synthetic and oil refinery data. Feedback from experts in the refinery data is documented including the comparison to dependency diagrams elicited from the experts. Some resultant explanations are also included that are based on networks learnt from variables grouped using the method in Chapter 4 in order to gain an insight into the speed of the overall process. This chapter is based on the work in [Tucker2001b].

Chapter 6 looks at the problem of offline learning of DBN models for explanation with changing dependencies. An extension of the representation from Chapter 4 is outlined which will ensure transparency and two methods are compared for learning the structure at the same time as segmenting the data into regions of different control structure. In addition, a new tool is introduced known as the Dynamic Cross Correlation Function (DCCF) which is used to analyse the datasets. Synthetic data shows very good results when compared to original structures and segmentations. The oil refinery data is compared to the DCCF analysis with very encouraging results. Sample explanations are included to show how the method has also

learnt more complex dependencies that are not evident in the DCCF analysis (which can only find pair-wise dependencies).

Chapter 7 concludes all of the results documented within this thesis and raises several problem areas for the methods as well as ways to overcome them. Various interesting areas for future work are also discussed.

2 Background

Within this chapter, the datasets that are used within this thesis will be detailed and their characteristics outlined in section 2.1. Various well-established dynamic models of data will then be documented in section 2.2 and finally a summary of the desirable features of each model is documented in section 2.3.

2.1 The Datasets

In this section, methods for generating datasets that are used throughout this thesis are described as well as characteristics of the real-world oil refinery dataset.

2.1.1 Vector AutoRegressive Data

The VAR process is a statistical multivariate time series model (see section 2.2). The VAR model has a wide variety of applications ranging from medical domains [Swift99a] to economic domains [Chatfield89]. Using VAR models of varying order, *M*, and dimensionality, *N*, 1000 time points of data have been generated (n = 1000) for training and testing DBN models in forecasting and explaining. One dataset is used in this chapter to show how a DBN can be used to model VAR process data and in Chapter 4, a number of VAR process generated datasets are used in order to test our MTS variable grouping algorithm.

2.1.2 DBN-Generated Data

Datasets with varying dimensionality, *N* and maximum time lags, *MaxT* have been generated using hand-coded DBNs. This requires imposing both structure and conditional distributions

upon a network. These hand-coded DBNs were used with an inference algorithm to generate data where the length of the MTS, *n*, was 1000. Stochastic simulation, which is described in detail in chapter 3, was the chosen method for inference. The algorithm for generating the data is given in Algorithm 2.1 and is graphically illustrated in Figure 2.1.

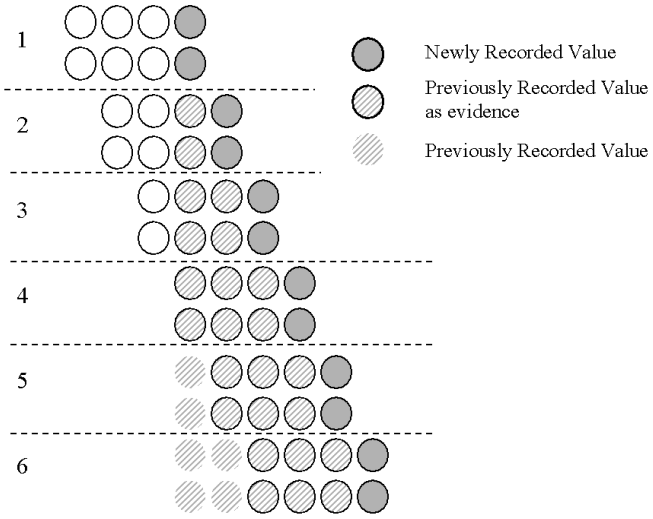


Figure 2.1. The Process of Data Generation using Stochastic Simulation on a hand-coded DBN. The Diagram shows Six Iterations on a very small network with *N*=2 and *MaxT*=4. The links are not shown.

```
1   Apply inference (e.g. Stochastic Simulation) given no
    information upon the DBN.
2   Record all values at current time slice t
3   For i =1 to n-1
4       Time-shift the DBN so that the values at time t are
        now at time t-1
5       Apply inference using any recorded values at time < t
        as evidence
6       Record all values at current time slice t
7   End For
```

Algorithm 2.1 - Generating Data using Stochastic Simulation

2.1.3 Oil Refinery Data

The real-world data used in this and later chapters is generated from an oil refinery in Grangemouth on the Firth of Forth in Scotland. The entire plant produces 34 Mb of data per day. The process focussed on in this thesis is from an Absorber / Stripper fractionating column within a Fluid Catalytic Cracker (FCC) and has approximately 300 measured variables. The measurements include temperatures, pressures, flow rates, and controller set points and modes (controllers are explained in the next paragraph). These measurements are taken every minute and are stored on the *PI-system* which is a set of server and client-based software programs designed to fully automate the collection, storage and presentation of plant data. Data has been downloaded from the *PI-system* into a series of Excel files. The original data is compressed, that is, only when a measurement 'moves' past a given threshold will the time stamp and process measurement be stored. To convert this data into a multivariate time series involves an algorithm which increments the time by a fixed amount and sets the measurement equal to the interpolation between the previous time stamp and the next time stamp.

Many of the variables are controlled. A controlled variable is one whose movement is being directly manipulated, sometimes manually or sometimes automatically by a controller. A controller has a Set Point (SP) which is a value that a controlled variable is manipulated to try and follow. The output (OP) from a controller will be used to reduce the difference between the actual value of the variable and the value of its set point. For example, the output from a controller that is trying to lower the value of a pressure variable will cause a valve to open more and, therefore, lower the pressure. There are different modes a controller can be in:

AUTO: where a controller automatically produces an `Output'. The output is used to determine an action which will rectify any inconsistency between the actual process value and its setpoint. Essentially, it is a closed loop, feedback system like a thermostat. For example, a pressure variable dropping below its set point causes the controller's output to decrease. This in turn, causes a valve to close slightly, resulting in an increase in the variable's pressure.

MANUAL: where a variable is being controlled, manually, by a process engineer to follow its set point. For example, the engineer is directly controlling a valve to increase the flow rate of a variable which has dropped below its set point.

CASCADE: where the set point of a controller is determined from the output of another controller. A collection of controllers can be set up like this for complex interactions between variables.

The mode, output and set point are all measured variables in the data set. However, some control information is not. That is, there is other controller information, specifically for use in multivariable control situations (known as Robust Multivariate Predictive Control Technology - RMPCT), which is not included in the data.

The decompressed data is in the form of a time series sampled every minute over approximately 300 variables. That is $60 \times 24 \times 300 = 432000$ measurements in a day. The dataset contains approximately 45000 minutes of data which is about one months worth, although an entire year's data is available in compressed format. The variables vary in type. Measurements are real values and state variables are discrete. The time it takes for one

variable to have an effect on another varies over time and between variables. It is not likely to be more than two hours but will often be between 0 and 60 minutes. The dependencies between some variables may change over time due to the change in operation of the refinery unit (how the unit is being controlled). This means that two variables may be very highly dependant for some time and then, suddenly, become uncoupled and behave independently. There is a very small percentage of data missing where, for example, a measurement or calculation fails. This accounts for less than 1% of the data.

2.1.4 Discretisation

Prior to learning the structure of a DBN we must discretise any continuous data into a number of *states*. There are many different methods for doing this. We can discretise the data based on patterns that are found in the data. For example, Dynamic Time Warping (DTW), [Berndt96], offers a way of 'stretching' a section of a time series along the time axis in order to match the data to a template. Wavelet Decomposition [Bakshi94] can be used to split the data into sections that can then be matched to templates using simple neural network pattern matchers. Alternatively, we can simply discretise the data into states based on their relative values. For example, range based discretisation involves breaking up the state space for each variable into equally sized ranges from the minimum to the maximum value. Figure 2.2(a) shows the ranges for each of four states using range based discretisation and also the resultant discretised data. Frequency based discretisation, on the other hand, involves dividing the state space into ranges of varying sizes so that the frequency of each state within the data is equal. Figure 2.2(b) shows the ranges for frequency based discretisation and the resultant discretised

data. [Friedman96] tries to improve the quality of data discretisation further by investigating a method for learning BNs and discretisation policies simultaneously.

(a)

(b)

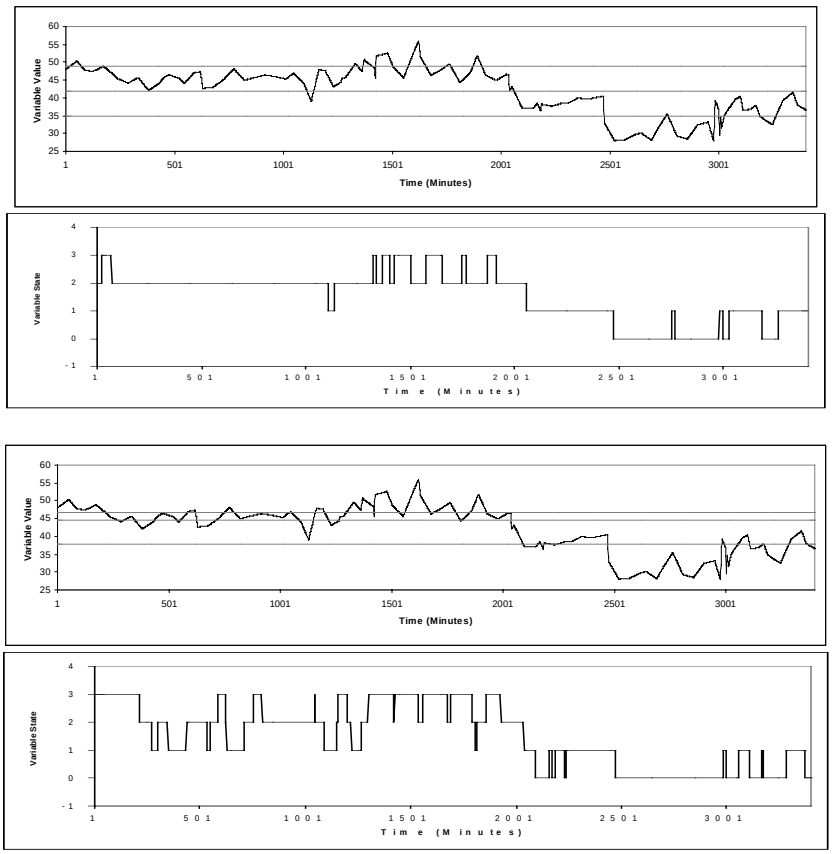


Figure 2.2. (a) Range Based Discretisation and (b) Frequency Based Discretisation

For the remainder of this thesis we will concentrate on the frequency methods of discretisation as in general it has been found that the frequency based discretisation produces DBNs that are more expressive than the range based method. This is probably due to the resultant data maximising the frequency of different combinations of instantiations. For example, the range based discretised data in Figure 2.2(a) contains only nine percent of cases where it is in state 3. This will result in networks that have been learnt on a smaller sample of many of the instantiations where this variable is in state 3. In contrast, the frequency based

data in Figure 2.2(b) contains precisely twenty five percent of the data for each of the four states.

2.2 Dynamic Models of Data

There are many different paradigms that are capable of modelling dynamic domains from classic statistical methods such as the vector autoregressive model to the relatively new dynamic Bayesian network architecture. In this section, some of these models are introduced and their suitability for tackling the problem of automatic explanation of MTS is discussed. This section introduces various different AI and statistical methods for modelling MTS and the suitability of these are summarised in section 2.3.

A Multivariate Time Series, *A*, with *N* variables of length *n* is to be modelled. Relationships between two variables a_i and a_j can exist over varying time lags where $a_i(t)$ represents variable a_i at time point *t* and $a_i(t_s,t_f) = \{ a_i(t_s) \ldots a_i(t_f)\}$.

2.2.1 Vector AutoRegressive Process

A commonly used statistical method for modelling MTS is the Vector AutoRegressive Process [Lutkepojl93], usually denoted as VAR(*M*) for a model of order *M*, as defined in Equation 2.1.

$$A(t) = \sum_{l=1}^{M} V_l \cdot A(t-l) + \varepsilon(t) \tag{2.1}$$

where $A(t)$ is the next data vector of size *N* (the number of variables in the model), V_l is an $N \times N$ coefficient matrix at time lag *l*, and $\varepsilon(t)$ is an *N*-dimensional zero mean noise vector at time *t* (usually Gaussian). The value of each element in V_l is usually a bound real number.

$A(t)$ is often assumed to have zero mean over the entire sample length $t = 1...n$, i.e. $\sum_{t=1}^{t=n} a_i(l) = 0.$

The standard statistical methods for fitting a VAR process to a set of data often consist of two steps: order selection (determining a suitable M) and parameter estimation (calculating the matrices V_i from the data). Order selection is commonly performed through the use of information theory based metrics such as AIC (Akaike's Information Criterion) [Lutkepojl93]. The standard methods for parameter estimation include *Maximum Likelihood* (ML) methods, the *Yule-Walker* equations method, and the *Least Squares* method.

It can be seen that this type of model will not really be suitable for automatically generating *explanations* for two reasons. Firstly, the matrices which form the model are not at all easy to interpret in that the parameters within them go through various complex matrix operations in order to forecast future values. Secondly, the construction of such models is not *automatic*, requiring order determination and parameter estimation. It should be noted, however, that certain procedures are being developed in order to try and achieve this [Swifty99a].

2.2.2 Transfer Functions

Transfer functions are used to relate an output MTS variable to one or more input MTS variables. The relationship between input and output is modelled through the linear filter in Equation 2.2 where $v(B) = \sum_{j=-\infty}^{\infty} v_j B^j$ is the transfer function, [BoxJenkins76] and $\varepsilon(t)$ is independent noise.

$$a_i(t) = v(B)a_j(t) + \varepsilon(t) \qquad (2.2)$$

The coefficients in such models are referred to as impulse response weights. Figure 2.3 illustrates how the transfer function is used to modify an input signal and a noise signal to generate the output time series. In order to build Transfer Functions based on data, the Cross Correlation Function (CCF) can be used.

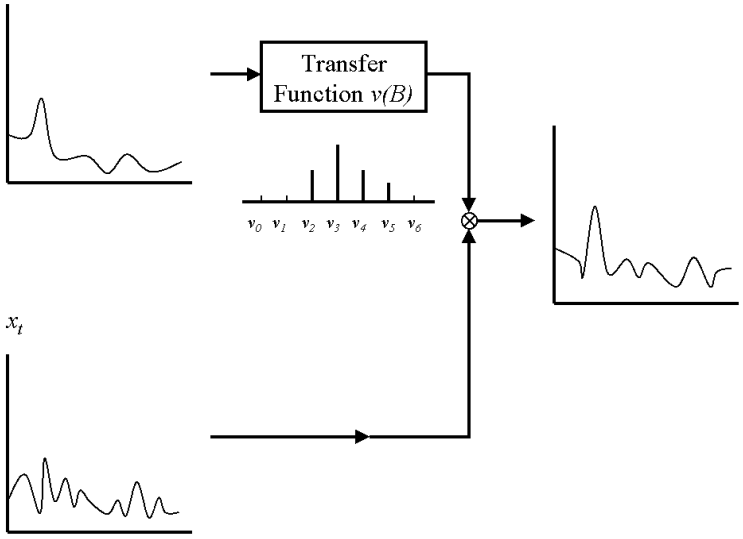


Figure 2.3. A Dynamic Transfer Function.

The CCF is a useful measure of strength and direction between two variables and is directly proportional to the impulse response weights. It is generated by calculating the correlation coefficient such as Pearson's [Pearson02] across two time series variables for varying time lags. Therefore, the variables are time-shifted up to some maximum time lag in both directions and the correlation coefficient calculated on these data. There are various correlation coefficients that can be calculated. Two of the most common are Pearson's Correlation Coefficient and Spearman's Rank Correlation. Spearman's Rank Correlation is defined in chapter 5 where it is used to group variables in an MTS. The CCF (using Pearson's) is calculated as follows:

$$\rho_{a_i a_j}(l) = \frac{\gamma_{a_i a_j}(l)}{\sqrt{\gamma_{a_i a_i}(0)\gamma_{a_j a_j}(0)}} \quad 2.3$$

This function makes use of the Cross Covariance Function which is calculated as follows:

$$\gamma_{a_i a_j}(l) = Cov(a_i(t,n), a_j(t+l,n)) \quad 2.4$$

$$Cov(X,Y) = E[(X-\mu_x)(Y-\mu_y)] \quad 2.5$$

where *Cov(x,y)* returns the covariance of *x* and *y*. See Figure 2.4 for an example CCF between two variables a_i and a_j. It can be seen from this that there is a strong positive correlation from variable a_j to variable a_i with a time lag of 6. The CCF will be used as an analysis tool in several chapters of this thesis.

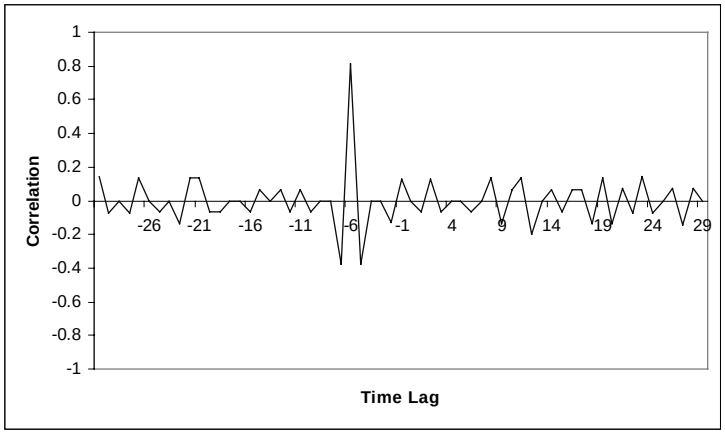


Figure 2.4. A CCF from variable a_i to Variable a_j with maximum time lag of 30 time slices. Notice the strong correlation for a_j to a_i with a time lag of 6.

In the context of automatically explaining MTS, the transfer model's biggest drawbacks are the same as the VAR model - difficult to interpret parameters, and impulse response weights that require a procedure of estimation (including an analysis of the CCF) and diagnostic checking, preventing automatic model building.

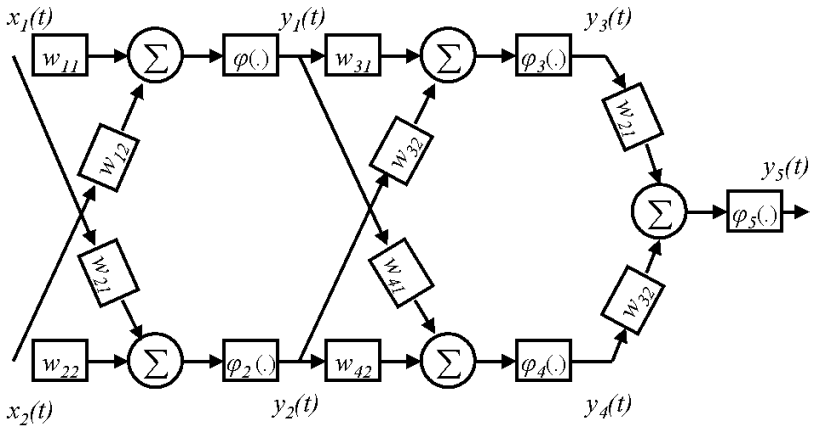
2.2.3 Time Delay Neural Networks

The Time Delay Neural Network (TDNN) [LangHinton88, Waibel89, Haykin94] offers a method for modelling MTS data using a multilayer perceptron architecture that encodes each neuron as a finite impulse response filters (FIR). This means that weights are computed for neurons over different time delays in order to capture the dynamic relationships between input and output variables. The general architecture of a TDNN and the FIR synaptic connections are shown in Figure 2.5(a) and 2.5(b). It can be seen that the output signal from synapse *i* is calculated according to the convolution Equation 2.6 below where *t* is discrete time.

$$s_{ji}(t) = \sum_{l}^{M} w_{ji}(l) x_i(t-l) \tag{2.6}$$

Unlike the previous statistical models, this paradigm should be very good for automatically learning accurate models from MTS using a specifically designed back-propagation scheme [Wan90]. However, its major drawback (as is the case with most neural network models) is that it is inherently black box in nature. In other words, however well it can model the data, the intricacies of the model will be of little use to a user who is interested in understanding the discovered relationships within the MTS. The complex relationships within the data will be hidden within the weights of the links between neurons.

(a)



(b)

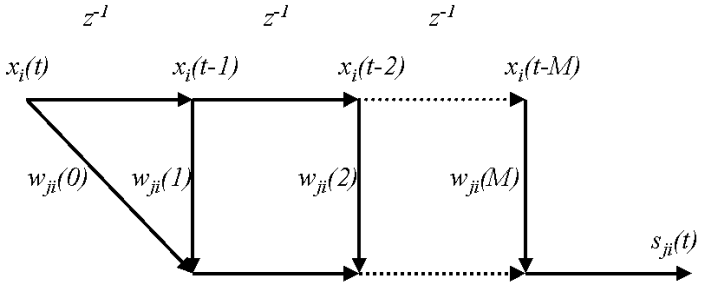


Figure 2.5. (a) The Time Delay Neural Network Architecture and (b) the FIR synaptic connections.

2.2.4 Hidden Markov Models

The Hidden Markov Model (HMM) [Rabiner1989] is a stochastic model which encodes the probability distribution over a sequence of observations by storing their conditional distributions given some discrete hidden state of the system. This unmeasured hidden state is conditioned upon the hidden state at the previous time slice. Based on the Markov property, the observations at each time slice are independent of one another given the hidden state of the system and the distribution at each hidden state can be calculated using only the information from the previous time slice. Figure 2.6 displays a HMM in graphical form.

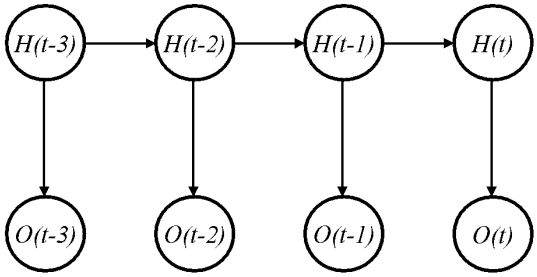


Figure 2.6. The Architecture of the Hidden Markov Model

It can be seen that this model requires two transition probability matrices. Firstly, the transition matrix between hidden nodes over time slices (e.g. between *H(t-2)* and *H(t-1)*, and secondly between the hidden node and the observed node (e.g. *H(t)* and *O(t)*). These transition matrices are the same over all time points - they are time invariant. In order to learn the parameters for such models from data, an Expectation Maximisation (EM) algorithm [Dempster76] is employed which is described later in this chapter with respect to DBNs with hidden nodes. HMMs have been used successfully in many applications, probably the most well known being speech recognition [Rabiner93].

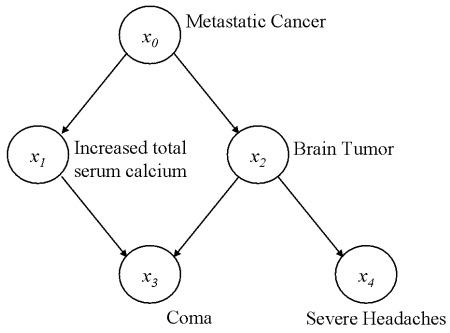
The HMM offers an ideal method for modelling MTS and many algorithms exist for parameterising models based on data. Whilst the parameters are essentially probabilities and should be therefore relatively easy to understand, the sheer size of the matrices that would be required for larger problems may make the paradigm unfeasible.

2.2.5 Dynamic Bayesian Networks

The Dynamic Bayesian Network (DBN) [Dagum95, Friedman98a, Gharmani98] is an extension of its static counterpart whereby a domain is modelled using a DAG and conditional probability tables. Bayesian Networks (BNs) offer a method for modelling domains probabilistically. They allow us to store the joint probability distribution of a domain by making conditional independence assumptions about variables. A thorough discussion of BNs can be found in [Pearl88, Neapolitan90]. As was described in BN consists of the following:

1. A graphical structure, S_h. This is made up of a set of N nodes, X, representing variables within the domain and directed links between them representing conditional dependencies between a node, x_i, and its parents, π_i. The structure must form a Directed Acyclic Graph (DAG). There must not exist a *directed* path from any node to itself in order to preserve this directed acyclic property.
2. A set of conditional probability distributions. Each node is assigned a Conditional Probability Table (CPT) which determines the probability of that nodes different possible instantiations given the instantiations of its parents, $p(x_i \mid \pi_i)$.

(a)



(b)

$P(x_0)$:	$P(+x_0) = 0.2$	
$P(x_1\|x_0)$:	$P(+x_1\|+x_0) = 0.8$	$P(+x_1\|\neg x_0) = 0.2$
$P(x_2\|x_0)$:	$P(+x_2\|+x_0) = 0.2$	$P(+x_2\|\neg x_0) = 0.05$
$P(x_3\|x_1, x_2)$:	$P(+x_3\|+x_1,+x_2) = 0.8$ $P(+x_3\|+x_1,\neg x_2) = 0.8$	$P(+x_3\|\neg x_1,+x_2) = 0.8$ $P(+x_3\|\neg x_1,\neg x_2) = 0.05$
$P(x_4\|x_2)$:	$P(+x_4\|+x_2) = 0.8$	$P(+x_4\|\neg x_2) = 0.6$

Figure 2.7. A Bayesian Network for a 5 variable domain. (a) Nodes represent variables and links represent conditional dependencies between variables. (b) The CPT for each node.

For example, Figure 2.7(a) shows a BN for a small domain containing five Boolean variables along with their CPT in 2.7(b) (from [Pearl88]). Note that $+x_i$ signifies that x_i is true and $\neg x_i$ signifies that it is false. Also note that each variable's distribution must sum to one so, for example, $P(\neg x_o)$ is equal to 0.8. Given a BN such as that in Figure 2.7 we can retrieve the joint probability distribution by multiplying over the conditional probabilities based on the states of each node for each *atomic event*. An atomic event is a situation where every node ahs been instantiated to some value. To calculate the probability of such an event requires the Equation 2.7.

$$p(X) = \prod_{i=1}^{N} p(x_i \mid \pi_i) \tag{2.7}$$

For example the probability of the atomic event where x_o is true, x_1 is true, x_2 is false, x_3 is false and x_4 is true is as follows:

$$p(+x_0,+x_1,\neg x_2,\neg x_3,+x_4) = p(+x_0)\,p(+x_1 \mid +x_0)\,p(\neg x_2 \mid +x_0)\,p(\neg x_3 \mid +x_1,\neg x_2)\,p(+x_4 \mid \neg x_2)$$

$$= 0.2\times 0.8\times 0.8\times 0.2\times 0.6 = 0.01536$$

Given the joint distribution of a domain, we can calculate the impact of various observations upon other unobserved variables. Trivially, we could calculate the joint using Equation 2.7

and then manipulate the appropriate probabilities to calculate the impact of the evidence on other nodes. However, it would be more compact and useful if certain observations could be made about the domain, these entered into the network (which will involve setting the probability of a node taking a particular instantiation to 1) and then calculating the impact of the evidence on the distributions of unobserved nodes in the network. There are existing inference algorithms which are capable of doing this.

In Dynamic Bayesian Networks, nodes will represent variables at particular *time slices* (or positions within a MTS) represented by the letter *t*. Therefore, dependencies can exist between nodes over different time *lags*. These models can make use of inference algorithms in the same way as in static BNs. This will allow them to be used for forecasting and monitoring. For example, in Figure 2.8 a DBN representing five variables is shown where there are dependencies between nodes within the same time slice (known as contemporaneous dependencies) and between nodes in different time slices (non-contemporaneous links). By making observations at time slice *t*, we can use inference algorithms to see the effect upon other nodes at different time slices in the past and future.

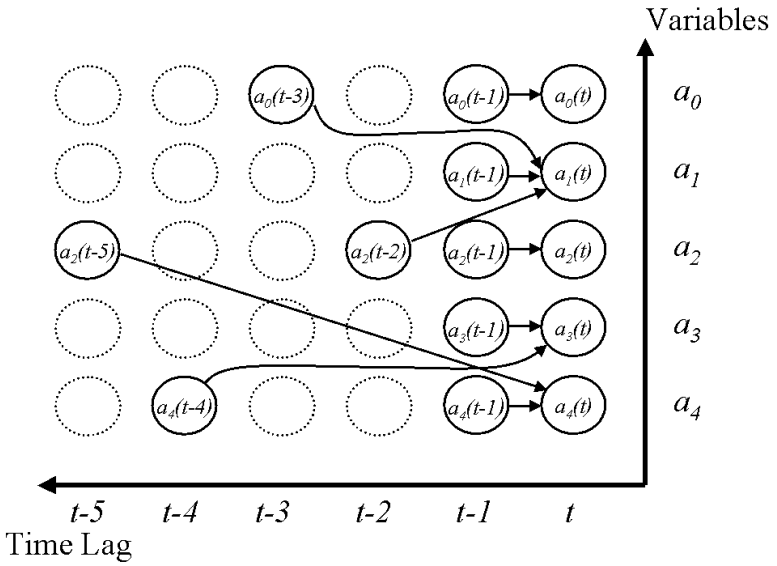


Figure 2.8. A Dynamic Bayesian Network (DBN) with 5 variables and 5 time slices

The graphical structure of a DBN offers an excellent way of displaying the relationships between variables over time. In the same way as with its static counterpart, nodes can be manipulated by entering observations and inspecting the impact of that evidence upon other nodes. What is more, whilst there is little research into learning DBNs from data, algorithms do exist for learning static models which can possibly be adapted.

2.3 Summary

Having looked at a selection of MTS models from both AI and the statistical fields, it has been decided that the model that fits the criteria most closely for the problem specification in Chapter 1 is the DBN. Table 2.1 summarises the features of each of the models discussed.

Feature	Vector AutoRegre-ssive Process	Transfer Functions	Time Delay Neural Networks	Hidden Markov Models	Dynamic Bayesian Networks
Transparent	✗	✗	✗	?	✓
Learn from Data	✗	✗	✓	✓	✓
Accurate Forecasting	✓	✓	✓	✗	✗
Scalability	✗	✗	✗	✗	?

Table 2.1. A Summary of some of the features of the MTS Models

Based on this summary, the DBN has been chosen to model the process data for generating explanations. Whilst forecasting will be limited to discrete states unlike the VAR process, the transfer function and the TDNN, the DBN can be learnt from data and remain transparent. In contrast to most other methods it has the ability to combine expert knowledge in a non-technical format with data. Many of the models will not scale well to high dimensions and this may include the DBN. Scalability issues, therefore, will have to be resolved.

Within this chapter, a comparison has been made of various different models for MTS data. Whilst accuracy in predicting data is not as good as other dynamic models such as the TDNN, the advantage of a DBN model of MTS data is its ability to explain events. The graphical structure is easy to interpret by making explicit the dependencies that have been discovered within the data. Additionally, inference within the DBN allows us to calculate the most likely set of events that have led up to the current observations. The discretisation of continuous variables will obviously result in loss of information and therefore reduce the

accuracy in operations such as forecasting. However, it will allow us to make more generalised statements about relationships between variables. For example, we can make a set of observations on a DBN and see what impact this has on nodes at previous time slices. In the next chapter some of the issues concerning BNs are outlined including some examples of explanations using inference and methods for learning such models from data.

3 Dynamic Bayesian Networks for Modelling MTS

In this chapter, firstly various issues concerning BNs (both static and dynamic) are documented. This includes a review of the work in learning such models from data, applying inference and modelling hidden variables. Following this review it is investigated how well a DBN can be used to model Multivariate Time Series such as Vector Auto Regressive (VAR) data and a simple process from the oil refinery dataset. Some preliminary data exploration experiments are undertaken in order to show how a DBN can be used to model the synthetic and real-world datasets. The remainder of this chapter is devoted to investigating how existing search methods for static BNs can be adapted to DBNs. These adapted algorithms are tested on several synthetic MTS. This work is taken from the research documented in [Tucker2001b].

3.1 Learning Bayesian Networks (Static and Dynamic)

Learning static BNs from data is an extensively researched area and a good review of related literature in the subject can be found in [Buntine96]. Most methods involve scoring candidate network structures, S_h, with some metric and applying a search strategy for exploring the space of different structures. The metric will reflect how well S_h fits a particular dataset, D. Methods for learning DBNs have not been researched quite so heavily and so this section will focus more on static network algorithms. Later in this chapter some of these methods will be adapted to learn DBNs.

As was pointed out in section 1.2, the explanations that will be generated from a BN that has been learnt from data will not necessarily involve direct *causal* influence [Pearl91, Heckerman97]. This is due to the fact that even if the optimal BN structure is discovered from data, several possible networks can represent the same conditional independences within data but portray a different *causal structure* (where links represent direct causal relationships). Different networks that encode the same conditional independence are said to be in the same *equivalent classes* [Pearl91]. For example, a network with just two variables, x_i and x_j and one link from x_i to x_j will have the same conditional independences as the same variables with the link from x_j to x_i. However, the causal dependencies have very different meanings.

3.1.1 Metrics

The relative posterior probability $p(D,S_h)$ is a commonly used metric and the logarithm of this is usually taken due to the extremely small values involved in computation. It is calculated from the log prior and the log marginal likelihood (see Equation 3.1).

$$\log p(D,S_h) = \log p(S_h) + \log p(D \mid S_h) \tag{3.1}$$

If we consider all network priors to be equal (assume a uniform prior), we can ignore the first part of Equation 3.1 and simply try and maximise the log marginal likelihood. This is the method by which Cooper and Herskovitz demonstrated the learning of BN structure from data [Cooper92]. The log likelihood was calculated as in Equation 3.2.

$$\log p(D \mid S_h) = \log \prod_{i=1}^{N} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{F_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} F_{ijk} \tag{3.2}$$

where r_i denotes the number of instantiations of a node x_i, q_i denotes the number of unique instantiations of the parents of node x_i, F_{ijk} is the number of cases in D where x_i takes on its kth unique instantiation and π_i takes on its jth unique instantiation, and $F_{ij} = \sum_{k=1}^{r_i} F_{ijk}$. Equation 3.2 can be calculated using Stirling's approximation for the logarithm of factorials [Stirling1730, Knuth69] to avoid computing extremely small numbers. This is calculated as in Equation 3.3 below.

$$\ln(y!) \approx y\ln(y) - y \tag{3.3}$$

In contrast to the maximum likelihood method we have briefly introduced, above, there is a family of metrics which try to minimise some information complexity measure. For example Minimum Description Length (MDL) which was used by Lam and Bachus, [Lam94], and Suzuki, [Suzuki96], to learn BN structure. The Description Length of a network is calculated in two parts. Firstly the DL of the network model, DL_{S_h} is calculated and summed with the DL of encoding D given S_h, DL_D. This is formalised in Equations 3.4 and 3.5.

$$DL_{S_h} = \sum_{i=1}^{N} \log(N) \times \left((r_i - 1) \prod_{j \in \pi_i} r_j \right) / 2 \tag{3.4}$$

$$DL_D = \sum_{i=1}^{N} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} -F_{ijk} \times \log\left(\frac{F_{ijk}}{F_{ij}} \right) \tag{3.5}$$

3.1.2 Search

The problem of searching for network structure has been shown to be NP-Hard [Chickering96a] and so various heuristic and approximate search methods have been explored.

The K2 Algorithm

In [Cooper93] a simple search technique, known as K2, was employed with the marginal likelihood which involved initialising a network with no links. With each iteration, a link was added that most increased the marginal likelihood of the network structure. This process was repeated until no link could be added which would improve the network score. This method required an ordering over the variables, was heuristic driven and deterministic, and therefore prone to suffer from local maxima in the search space.

Branch and Bound

Branch and Bound [Suzuki96] applies a recursive search whereby the exhaustive search is limited through the calculation of a minimal bound. This bound will determine whether any further evaluation is necessary along the current branch of the recursive search. If the current score is better then the bound calculated for the remainder of that recursive branch then no more calls to that branch are necessary. The technique will be sure to find the optimal solution as it offers a method of limiting the exhaustive search. It is however, only applicable to the MDL metric as no simple bound exists for the log marginal likelihood metric where a uniform prior is assumed.

Genetic Algorithms

Various Evolutionary methods have been investigated in order to try and overcome the ordering requirement and the huge search spaces involved. [Larranaga96] investigated the use of a Genetic Algorithm (GA) in order to search for structure. The GA was first introduced by Holland in 1975, [Holland95], and is a procedure for searching for the global optimum

through the application of certain operators upon a population of candidate solutions. The algorithm was designed to mimic the kind of processes which occur in the natural processes of evolution, where organisms evolve to solve particular problems in order to survive through the inherited characteristics of their parents.

```
1    Initialise random Population of size Popsize
     For i=1 to Generations
2        Select Parents from Population
3        Generate Children from Selected Parents using
         Crossover according to CrossoverRate
4        Apply Mutation to random Individuals according
         MutationRate
5        Add Children to Population
6        Remove the least fittest individuals until
         Population is of size Popsize
7    End For
O/P  The Fittest Individual in the final Population
```

Algorithm 3.1 - The General Genetic Algorithm

This search is applied to a *population* of individuals which are known as *chromosomes*. Traditionally, these chromosomes are made up of a string of binary digits which represent possible solutions to the problem at hand. The search is implemented in a GA through the use of recombination and mutation operators, the most common being Crossover and Mutation although hundreds of different variations exist. Recombination operators cut and splice bits of *parent* chromosomes to create new *children* bit strings. Mutation operators update current chromosomes by making random changes to a number of bits within a chromosome. The operators are applied repeatedly in an iterative process where each iteration is known as a *generation*. At the end of each generation a process known as *survival of the fittest* is applied which involves reducing the size of the population to its original value before recombination increased it. Chromosomes are deleted from the population based on their *fitness* so that only

the fitter individuals remain for the next generation. The General GA process is defined in Algorithm 3.1.

Larranaga used the standard genetic algorithm to search for BN structure using a binary chromosome to represent network structure by converting the string to an $N \times N$ connectivity matrix, C where each binary element c_{ij} represents a link from node x_j to node x_i if equal to 1. Therefore the network in Figure 2.7 would be represented by the following matrix:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$ which is represented as a chromosome by the string

$$(00000\,10000\,10000\,01100\,00010)$$

Crossover involves deciding upon a random position on the chromosome and using this position to split each parent chromosome and generate to children by attaching the first half of one parent to the second half of the second parent and vice versa (see Figure 3.1).

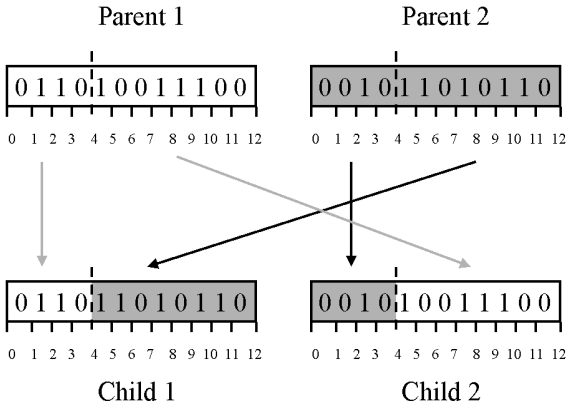


Figure 3.1 - The Crossover Operator applied to two binary chromosomes where the crossover point = 4.

Unfortunately, using the binary representation with crossover can result in children that do not represent legal structures (two legal acyclic parents can generate a child that represents a cyclic network). We say that the crossover operator is not a *closed* operator.

Mutation involves randomly altering single bits within the chromosome (based upon the parameter Mutation Rate) so that a one becomes a zero and a zero becomes a one. Larranaga's representation means that mutation is also not a closed operator as it can result in cyclic networks.

Survival of the Fittest (SOF) involves selecting individuals to go through to the next population based on their fitness. There are many different forms of SOF but the simplest is to reduce the swollen population (from the addition of children) to its original size by removing the surplus individuals with the lowest finesses.

Using the GA in Algorithm 3.1 with the representation and operators described above, as well as repair operators which converted invalid chromosomes into valid ones representing acyclic structures, Larranaga showed how BNs could be learnt using GAs without having to impose an order on the nodes (unlike K2).

Evolutionary Programming

```
1    Initialise a random Population of size Popsize
2    For i=1 to Generations
3        Generate a child for each member of Population using
         Mutation
4        For p=1 to Popsize
5            Select individual p from Population and q other
             random individuals
6            For each random individual with a fitness less
             than the fitness of  individual p add one to its
             score
7        End For
8        Randomly Mutate all individuals in the new
         population and remove any that are not Legal
9        Select the Popsize individuals with the highest
         scores to recreate the next population
10   End For
O/P  The Fittest Individual in Population
```

Algorithm 3.2 - The General Evolutionary Program

[Wong99] used the MDL principle with an Evolutionary Program (EP) and some modified operators in order to improve the speed of convergence of the standard GA. An EP differs from a GA in several ways [Baeck96, Fogel95]. Firstly, the emphasis is upon mutation as opposed to recombination, where parents are selected to go forward to the next generation after a mutation operator is applied. It is typically applied to real valued domains though not restricted to only these as any representation can be used that is not necessarily real or binary. *Tournament Selection* [Baeck93] is the most common method of selecting parents for mutation in an EP. This involves taking each individual and comparing its fitness to a selection of *q* other randomly chosen individuals in the population. The individual in question then is scored based on how many of the *q* other individuals have a fitness lower then its own. The General EP with Tournament selection is shown in below in Algorithm 3.2.

Wong's modified operator made use of some pre-processed information, whereby the Description Length of all possible individual links was calculated and used to bias the addition and removal of links in candidate networks. This heuristic has been used several times before. For example, [Chow68] and [Sahami96] both utilise a similar approach whereby the Mutual Information of all pairs is calculated and used to order them. The former applied the heuristic to learning trees and a best first search was adopted in the latter paper.

Turning now to dynamic Bayesian networks, there has been relatively little documented work in investigating efficient algorithms for learning models. We will look at adapting existing algorithms for static BNs to learn their dynamic counterparts at the end of this chapter (section 3.5). Friedman has described simple hill climbing techniques to independently learn models involving the contemporaneous links between time slices and the non-contemporaneous links within a time slice [Friedman98a]. Dagum, [Dagum95], investigated the forecasting of MTS using DBNs which were learnt using classical time series analysis with maximum likelihood methods. These DBNs were used for prediction on the course of critical care patients and pointed out the disadvantages of classical methods which assume linear relationships amongst variables and normal probability distributions. In particular, he highlighted the problems associated with spurious and implicit dependencies which could result in overly connected networks and therefore slow or even intractable inference.

3.1.3 Spurious and Implicit Dependencies

Spurious correlations and implicit dependencies were highlighted as problems that classical approaches encounter. A spurious correlation is a dependency that appears to exist between

two nodes due to a common parent between the two nodes (see Figure 3.2(a)), an implicit dependency is one which appears to exist due to indirect causes (see Figure 3.2(b)).

(a) (b)

Figure 3.2. (a) Spurious Correlation denoted by a dotted line between the node representing $a_0(t)$ and the node representing $a_1(t)$. (b) Implicit Dependency between the node representing $a_0(t-1)$ and the node representing $a_1(t)$.

In contrast to classical methods, the dependencies between two nodes that are discovered in BNs can be determined as explicit (i.e. not spurious or implicit) if a set of nodes cannot be found such that when we instantiate them, we break the dependency. This is from [Pearl92]. It will be important to ensure that any dependencies are explicit in an explanation model in order to prevent misleading or incorrect explanations.

3.2 Inference

In this section we look at difference ways to perform inference in both static BNs and DBNs.

3.2.1 Inference in Static BNs

In order to compute the *posterior* distribution of a BN given some observations, we must apply an inference algorithm to propagate this evidence throughout the network. Kim and Pearl introduced an algorithm to perform inference in Polytrees [KimPearl83]. Polytrees (also

known as singly connected networks) are network structures that contain no loops irrespective of link direction. The algorithm involves a message passing system which updates the beliefs in each node based on evidence from the node's parents (the causal support) and evidence from the node's children (the evidential support). This algorithm falls down, however, when applied to multiply networks (networks with loops) because the messages passing around the system cause nodes to oscillate indefinitely around the loops.

Exact inference in multiply networks has been shown to be NP-Hard [Cooper90]. Therefore, different methods have been adopted to try and circumvent this problem. Some methods such as clustering and conditioning convert the multiply networks into polytrees; others use approximate algorithms such as stochastic simulation. Clustering involves merging nodes in multiply networks so that polytree inference algorithms may be applied. For example, the BN structure in Figure 2.7 (which is multiply) would have the offending nodes, B and C, combined into one. This would mean that a new node, B & C, would be formed using the combined distributions for the new CPT (see Figure 3.3(a)). Conditioning involves splitting a multiply network structure into several polytree structures based on instantiating a certain set of nodes. For example, by instantiating node A from Figure 2.7, we get a structure such as that in Figure 3.3(b) for each instantiation of A (i.e. one polytree where A=false and one where A is true).

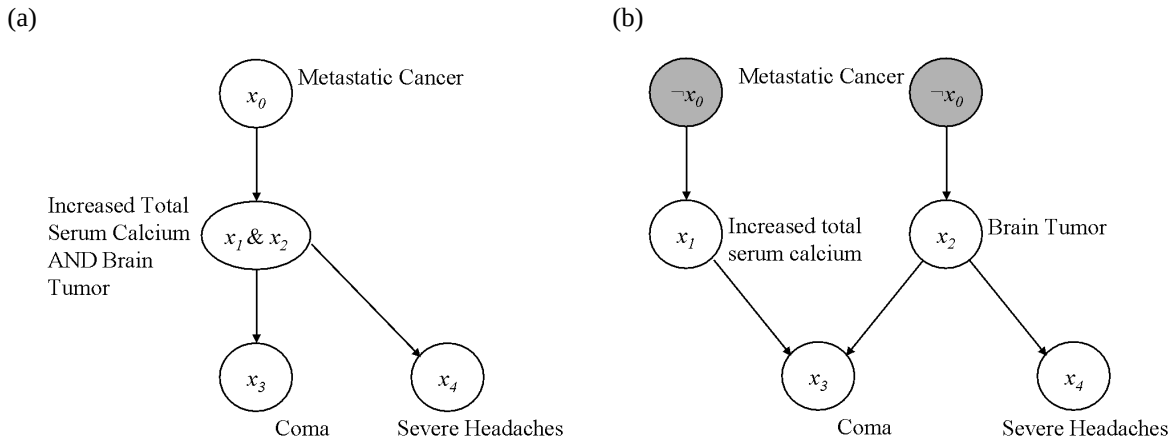


Figure 3.3. Inference in Multiply Networks by (a) Clustering and (b) Conditioning

Stochastic Simulation is a method to try and approximate the posterior distributions of a BN by running repeated simulations in order to generate multiple samples over the nodes. As the number of simulations increase, the frequencies of the states of each node occurring in the samples will approximate the exact posterior distribution. Logic Sampling as suggested by [Henrion88] handles the propagation of evidence by discarding all simulations where the observed nodes have been instantiated to states different to those observed. This can result in many simulations being discarded, especially as the number of observations increases. Pearl suggests a method to overcome this problem by clamping the observed nodes to their respective states and applying a two-step algorithm. All unobserved nodes are sampled given the other instantiations in the BN and then a biased random number generator is used to select the next state of that node.

This algorithm is outlined in Algorithm 3.3 (from [Pearl88]). Let w_i denote all other variables in a BN except variable x_i. The algorithm calculates $p(x_i \mid w_i)$ by local computation using the Markov blanket of x_i. The Markov blanket of a node consists of its parents, its children, and

the parents of its children. For example, the Markov blanket of node x_2 in Figure 2.7 will consist of all the other nodes in the network { x_0, x_1, x_3, x_4 }, the Markov blanket of x_0 will be the set { x_1, x_2 }, and the Markov blanket of x_4 will be { x_3 }.

```
1  Set all observed nodes to their appropriate states
2  Set all unobserved nodes to random states
3  For s = 1 to Sims
4      For i=1 to N
5          Calculate  p(x_i | w_i)
6          Generate the next state of x_i using a random number
           generator biased to the distribution computed in
           step 4
7      Next i
8  Next s
9  Take the average conditional probability calculated from
   step 4 as the posterior distribution for each node
```

Algorithm 3.3 - Stochastic Simulation

Another method of inference aims to generate the most probable configuration of values for all unobserved nodes rather than their posterior distribution. We cannot simply choose the state with the greatest probability for each node in order to calculate this. For example, if we were to imagine that nine people entered a lottery, eight buying one ticket each and one buying two, and we modelled this as a BN where each node represented a person with two states, win and lose. The posterior distribution should be for each of the eight with one ticket [0.1, 0.9] for the states win and lose, respectively. This posterior shows that the state with highest probability for all contestants will be lose. Even the person who buys two tickets will have a posterior distribution [0.2,0.8], and so will be likely to lose. However, if we want to calculate the Most Probable Explanation (MPE) [Pearl88], this would be the person with two tickets instantiated to win and the remainder to lose. In the context of automatically explaining MTS, this procedure may be useful. However, standard belief propagation is less

prone to influences from irrelevant information. MPE results can also be deceptive in situations where even the most probable explanation is very unlikely. In general using standard propagation to calculate posterior probabilities is better in prediction and diagnoses situations and for this reason it will be used in the remainder of this thesis.

3.2.2 Inference in DBNs

Inference in DBNs is almost identical to that of static BNs. Given a set of observations, algorithms can be employed to calculate the posterior distributions over unobserved nodes, both in previous time points or into the future. Standard inference algorithms can be used such as stochastic simulation which was discussed in the previous chapter. What is more, the distributions of nodes which are outside the initial time-scale of the original DBN can be computed. This involves a process called *scrolling*. For example, consider the DBN in Figure 2.6. If observations can be made about variables in the system, we can apply inference to gain the distribution of other unobserved nodes. The entire DBN can then be shifted forward one time slice so that variable $a_0(t-1)$ becomes $a_0(t)$. This is known as *scrolling* the DBN forward [Dagum92]. Essentially all nodes that were previously conditioned upon nodes at previous time points have their distributions converted so that any conditionals become priors. Having done this, inference can be re-applied using the distributions that were calculated from the previous inference. These are used as observations to update nodes in the new time slice and the process can be repeated to further time slices in the future. In the same way, the DBN can be shifted *back* in time to infer distribution in previous time slices.

If a DBN is used to monitor a system, the inference method described, can be vastly improved upon. This is due to available sensor information that is not being used to update the current state of the model in any way. To demonstrate this, consider tracking an object over time within 2D space using a weak state transition model such as random walk, but an almost error prone sensor. Figure 3.4 (taken from [Kanazawa95]) shows how the model evolves so that the possible states of the system spread out randomly over the space whilst the object follows its particular trajectory. Most of the projected states bear no relation to the actual position of the object and so only a very small sample of projections will be used to update the estimated next state, resulting in large errors. This type of prediction is not taking into account the fact that the true state of the system (the actual position of the object) is known with very little error at each time point. Kanazawa, Koller and Russell have researched into inference in stochastic processes [Binder97, Boyen98, Koller97] and experimented with algorithms for resolving this situation. In particular, a Survival of the Fittest (SOF) algorithm which works by only allowing samples that are similar to the current state of the system to be propagated [Kanazawa95].

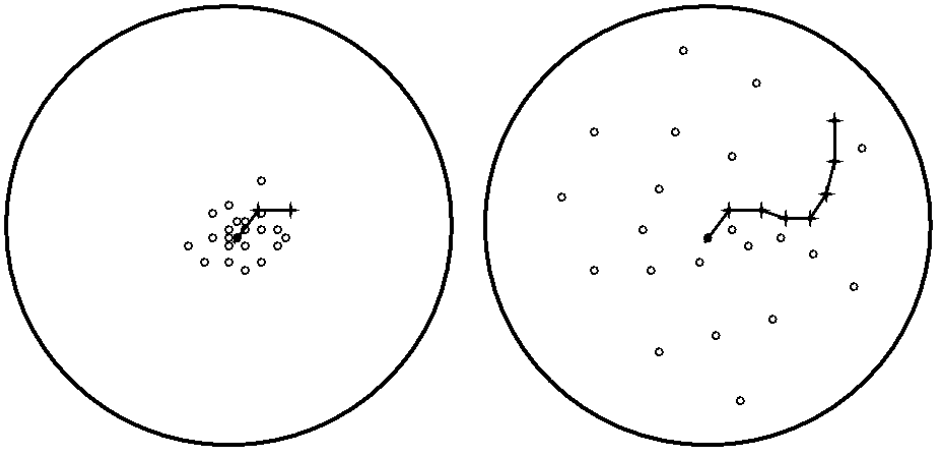


Figure 3.4. A Sample Monitoring Problem. Taken from [Kanazawa95].

This problem does not arise in non-monitoring problems such as projection because the sensor information at future time slices is not available. In explanation, where the DBN is shifted backwards to explain current events, the data will obviously be available. However, the question of whether to use data to update the inference process has not been investigated within this thesis. It was decided that a standard stochastic simulation method would be used which was introduced by Pearl, and is essentially the Algorithm 3.3.

Stuart Russel et al. have been investigating DBNs, including the application to modelling car driving behaviour [Forbes95]. In particular, they are attempting to develop a system for driving a car autonomously using a DBN as one of the central models.

3.3 Modelling Hidden Variables

A feature that is very characteristic of process data in general is that dependencies between variables can change over time. For example, in the oil refinery dataset the behaviour of some variables will be dependent upon the control engineer responsible for that particular process. Different engineers will control certain variables in different ways such as reducing a particular pressure by lowering a temperature somewhere else in the plant or by opening an associated valve. Dependencies may also be affected by the product that is being maximised by the refinery at any particular time. Changes in dependency will have to be taken into account when learning models from MTS data, especially when the MTS is long and has, therefore, had more chance of experiencing multiple switches. There is an extensive literature in state space models that incorporate changing dependencies or 'switching states'. A review of this literature is included in [Gharamani99] which describes the different architectures that

have been explored for modelling MTS with switching states and introduces a method for modelling discrete and continuous dynamics based on the hidden Markov model. For example, [Shumway91] makes use of an architecture which uses a state vector that is independent of previous states and is restricted to switching only the observed matrices (O in Figure 2.6) as opposed to the state matrices (H in Figure 2.6) which was the case in [GordonSmith88]. The method adopted a pseudo Expectation Maximisation (EM) algorithm [Dempster76] for the earlier stages and a non-linear optimiser procedure to maximise likelihood at the latter stages. The algorithm was tested successfully on synthetic multiple target tracking data.

For BNs, changing dependencies can be modelled using hidden nodes which are not measured and, therefore, do not appear in the dataset. A good example of how the exclusion of a hidden node can cause overly complicated BN structures is shown in Figure 3.5. In order to capture the distribution of the structure in Figure 3.5(a) many more dependencies are required if no hidden variable is included (Figure 3.5(b)).

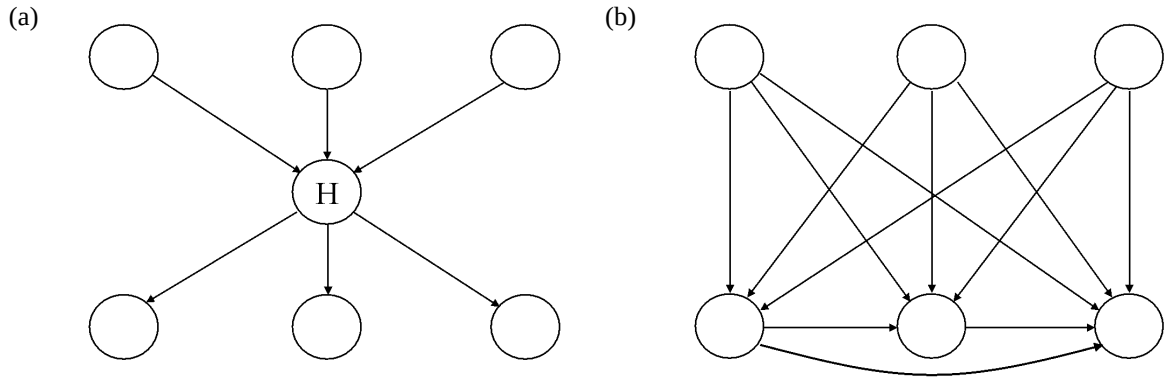


Figure 3.5. The use of a Hidden Node to simplify BN Structure. If the hidden node in (a) marked with an 'H' was not included in the model, the only way to capture all of the dependencies between the measured variables would be using the structure in (b).

Recently there has been a growing interest in learning BNs and DBNs with hidden nodes [Chickering96b]. Friedman, in particular, has developed an extension to the EM algorithm to learning network structure in the presence of missing data or hidden nodes. This algorithm, known as the Structural Expectation Maximisation (SEM) algorithm [Friedman97, Friedman98b] is described in general terms in Algorithm 3.4.

```
I/P  Initial Model with random structure S_h and random
     parameters except from observed variables, A
1    i = 0
2    Repeat until convergence or i > Iterations
3
4        E-Step
5        Improve model parameters through Parametric-EM given
         S_h and current parameters

6        M-Step
7        Search for structure that improves the expected
         score given the current parameters using a standard
         scoring metric such as log likelihood or DL

8        i = i + 1
9    End Repeat
O/P  Final Structure and Parameters
```

Algorithm 3.4 - The General SEM Algorithm

The E-Step is often implemented using an inference mechanism to calculate the probability of the hidden variable given the observed [Lauritzen92]. Friedman has proved that improvement in expected score results in an improved objective score of a BN model [Friedman98b]. The methodology has been tested on various datasets in order to learn BNs from datasets with one or more hidden variables. The biggest drawbacks concern the time-consuming process of calculating expected statistics during the M-Step and encountering local maxima which are common in this problem. Methods to overcome this include local

perturbations and an alternating SEM algorithm where the E-Step is allowed to progress for a number of steps or until convergence before any change is made to the structure, and a process known as deterministic annealing [Ueda95] which alters the likelihood's surface plot so as to allow the search to home in on the global maximum early on.

3.4 Preliminary Data Exploration

This section focuses on learning small networks from continuous data and tests the resultant DBNs by forecasting future values and comparing to actual observed values. The goal of this section is to identify the different stages involved in generating explanations automatically as well as any potential problems.

3.4.1 Modelling the Data

Frequency based discretisation was applied to both the synthetic VAR data and a 4000 minute section of the real-world controller data to generate four states over all variables. The structure is scored using the log marginal likelihood as a scoring metric and an exhaustive search strategy. The search involves adding links from one of the nodes at time lag greater than zero to a node at time lag zero. As the VAR data is a fourth order process, we search for links between all nodes up to a time lag of four and we search for all time lags up to five minutes for the controller data. The discovered DBN model can be used to predict states of variables in the future or to explain states of variables back in time through various existing inference algorithms. Here we apply stochastic simulation to the network in order to predict the following states of the variables given the previous observed states. This was applied to a new piece of data that the network was not trained on. We apply one step ahead forecast and

five-step ahead forecast on the oil refinery and VAR process datasets, respectively. We compare these forecasts to the actual data that was observed.

The parameters for the VAR process should determine the dependencies within the data. The structure, learnt from the first five hundred time points, shown in Figure 3.6 displays several dependencies between the variables with all links pointing from one of the three variables a_2, a_3 or a_4 at various time lags to all the variables at time lag zero. Figure 3.7 illustrates a five-step ahead forecast on a section of the remaining five hundred time points. The forecasts such as this one are fairly accurate, rarely predicting a state that is out by more than one.

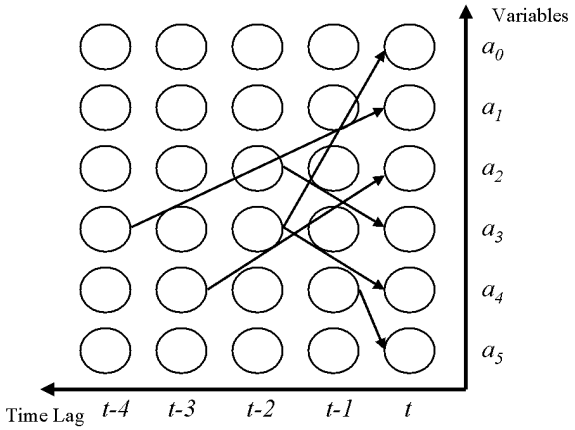


Fig 3.6. The DBN discovered from the VAR data

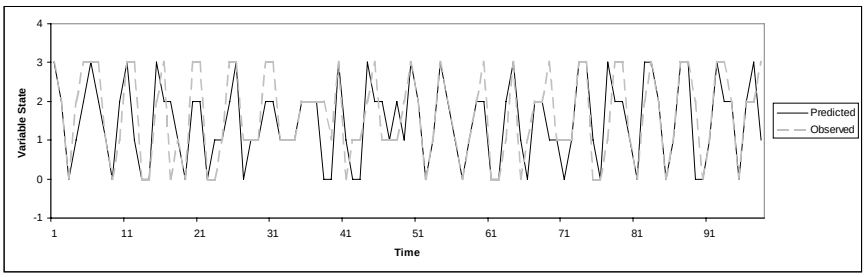


Figure 3.7. A five-step ahead Forecast on one VAR data variable using the discovered DBN

Next, a DBN is learnt from three variables relating to an automatically controlled flow rate within the FCC. The three variables relate to the actual recorded flow rate (PV), the set point of the flow rate (SP) which is the ideal rate of flow, and the output (OP) of the flow controller. These should be very closely related to one another as discussed in the previous section. The learnt structure, shown in Figure 3.8, highlights the dependencies that were discovered between the control variables. SP at time zero is determined by itself and PV one time slice previously, PV is determined by itself and OP in the previous time slice and OP is determined by itself and SP. This structure mirrors the sort of dependencies that we expected to find in this data (see the description of the behaviour of a controller in Chapter 2).

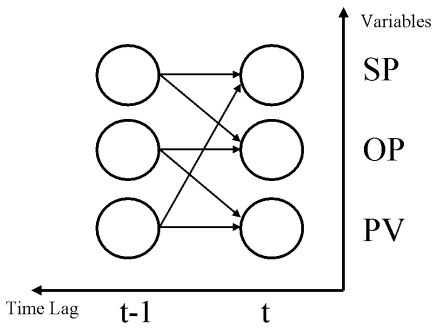


Figure 3.8. The DBN discovered from the Controller data

Figure 3.9 shows the one-step ahead forecast for the learnt DBN on the discretised FCC flow rate data. Once again as with the VAR data, the forecast is generally very good.

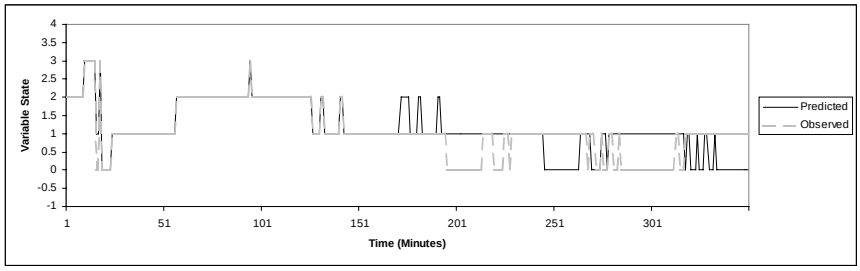


Figure 3.9. A Sample one-step ahead Forecast on PV using the Controller DBN

Given that the methods described in this chapter are limited to forecasting states of data rather than actual real-valued variables, the results are promising. They give a strong indication of the quality of the discovered DBN models. However, if we are to generate more accurate forecasts with real valued predictions, there are many other methods such as the TDNN introduced in Chapter 2 which will produce better results.

3.4.2 Generating Explanations

Figure 3.10 shows the posterior probabilities of variables in the VAR and Controller data given some observations. The x-axis represents the number of slices *back* in time from *t* and the y-axis represents the probability of a node being in state 2 or 3 (i.e. the higher two of the four states). Notice how the on the VAR process, some of the graphs have breaks. This is where there is no information that can be propagated to those variables at that particular time slice. Also notice on the controller data, the way that the shape of each graph is reflected in the other graphs back in time. For example, the sudden switch from low to high probability in PV is reflected by a similar behaviour in OP a few time slices previous and then in SP a few time slices previous to that.

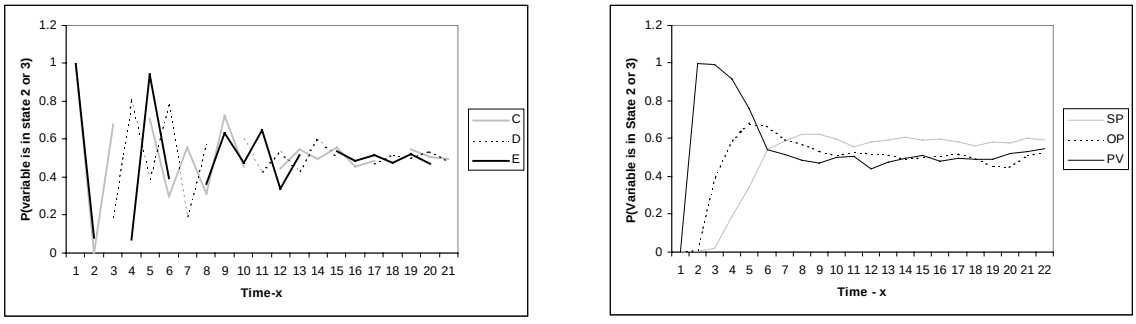


Figure 3.10. The Posterior Probability over three of the VAR Process Variables (a), and the Controller Variables (b), as the Explanation is generated back in time. Note the breaks in the lines in (a) where we have no information about a particular variable at that point in

time.

Figure 3.11 shows an example explanation that has been generated using the controller network. This is generated directly from the DBN structure where each node is replaced with its most probable state given the observations. Note that the explanation is formed from phrases such as 'PV in_state 3'. This essentially means that PV was between two limits depending on those used for discretising variable PV. It can be loosely interpreted as PV is 'high'. An explanation for 'PV in_state 0' (low) at the current time slice given 'PV in_state 3' (i.e. 'high') in the last time slice (as well as some other observations over the other variables) is shown in Figure 3.11. The arrows show the dependencies between events as found in the DBN structure and the probabilities relate to those found in Figure 3.10(b).

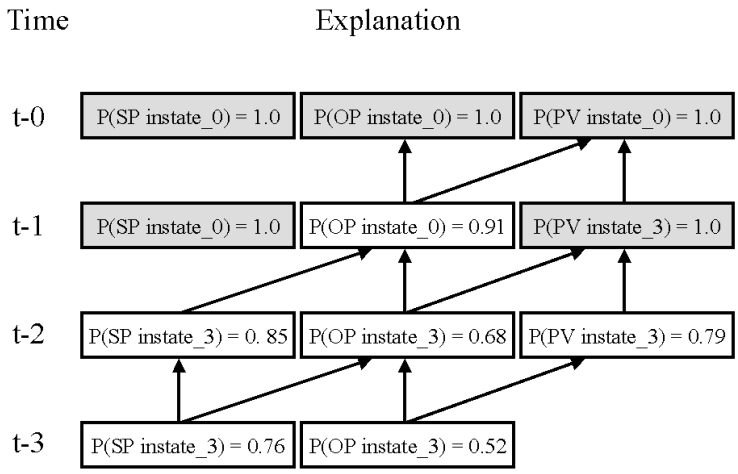


Figure 3.11. An Example Explanation using the Controller BN. Shaded boxes represent input to the network. Unshaded boxes represent possible causes for events.

The input observes that currently the PV is in state 0 but in the previous time slice it was in state 3. The inference back in time generates the probabilities over the three variables back in time. The output shows the OP changing from being in state 3 with a probability of 0.68 to

being in state 0 with a probability of 0.91, given the input. This can be interpreted as whenever PV is in state 3 and changes to state 0 in the following time slice, OP will have changed to state 3 one minute ago with 0.91 probability within the training data. Further inference, shows that given the distribution over OP at *t-1*, SP was in state 3 at *t-2* with a probability of 0.85.

3.5 Adapting Existing Search Strategies to Process Data

The existing methods for learning *static* BNs from data are many and varied and some were briefly discussed in section 3.1. In this section, a representation is introduced which permits some of these methods to be adapted to learn DBN structure. The different algorithms are compared and contrasted when applied to MTS from process data in order to see which methods are most suited to learning from large MTS such as oil refinery data. The learning curves are investigated of the K2 algorithm, a GA, an EP with knowledge guided mutation, and Branch and Bound. These are all tested with both log likelihood parameters and MDL on synthetic datasets. Note that experiments involving the stochastic algorithms (EP and GA) are repeated ten times and the average taken in order to give a clearer indication of overall performance.

Each algorithm has been adapted to the learning of DBNs and are described in detail below. The resulting curves are compared and contrasted. The goal of these comparisons is to identify any salient features that may be adopted in some hybrid algorithm for learning good networks rapidly from process data with large time lag.

3.5.1 Introducing a Representation for Dynamic Bayesian Networks

From now on the assumption is made that the process being modelled is being recorded frequently enough to allow us to exclude any instantaneous dependencies. That is our DBN will contains *no links within the same time slice* (also known as *contemporaneous* links). In many applications where data is recorded frequently, the assumption that all variables take at least one time slice to impose any effect on another may well be true. In other words it is impossible for there to be instantaneous relationships between variables - they all must take at least one time slice (if the time slice is small enough) to affect another. It should be pointed out that if some temporal order is put upon each link in this way, then the chances of a link representing a causal dependence are increased because many networks with the same equivalent class are removed from the set of possible candidate structures.

A DBN with only non-contemporaneous links can be represented by a selection of $N + |Q|$ nodes, where N is the number of variables at a single time slice, t, and Q is the set of nodes at previous time slices up to some maximum lag $MaxT$ ($|Q| \leq N \times MaxT$) where members of Q have a direct dependency on nodes at time slice t. We can use a list of triples to represent a possible network: (a_i,a_j,l) where a_i is the parent variable which is represented by a node in Q , a_j is the child variable and l is the time lag. Therefore, each triple maps directly to a link in the network. So for $N = 6$, $MaxT = 4$ and $|Q| = 6$, a list such as:

$$((2,3,2),(3,0,2),(3,4,2),(3,1,4),(4,2,3),(4,5,1))$$

would represent the DBN discovered from the VAR data in Figure 3.6. This notation is illustrated in Figure 3.12 for a DBN with N variables and $MaxT$ time lags.

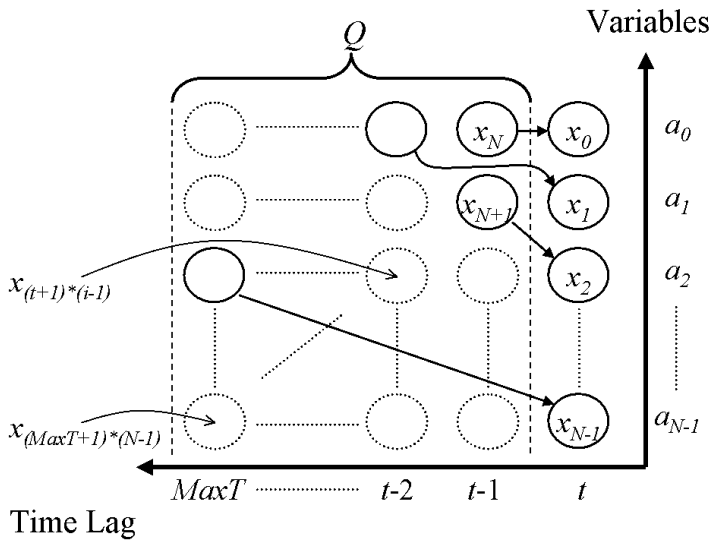


Figure 3.12. A Summary of the Notation used for the DBN Representation showing an example DBN with *N* variables, *MaxT* time lags and four links. Each parent is a member of *Q* and each child a node at time *t*.

The search space for this representation can be reduced because each node at time 0 and its set of parents will be independent of other variables at time 0. Therefore, we can search for the parents of each variable independently and the actual search space will be $N\left(2^{N.MaxLag}\right)$.

3.5.2 Adapting the Algorithms

We will now describe the algorithms that have been implemented and compared. These algorithms have been adapted to be used on DBN structures using the proposed representation.

For these experiments an operator, *Legal*, was defined which returned *true* if and only if

i) The number of parents did not exceed a parameter, *MaxBranch*, which was set as 3.

ii) There were no repeated triples within in each triple list.

K2 / K3 (adapted for DBN from [Cooper92] and [Bouckaert94])

This algorithm usually requires an ordering on the nodes. However, due to the assumption that is made based on ignoring contemporaneous links, this ordering can be ignored as all nodes will be ordered upon their time slice, t. It works by iterating through each node at time t and scoring the effect of adding all possible single parents to the current node. The parent that increases the score the most is then added to that that node's list of parents. This procedure is repeated until there are no parents that can be added to any nodes at time t, that will increase the network's score

```
I/P  Set of N+|Q| nodes, A (n×N MTS), upper bound on number
     of Parents, MaxBranch
1    For i=0 to N-1 (each node at t=0)
2        π_i = ∅
3        P_old = g(i,π_i)
4        Proceed = True
5        While Proceed
6            Let z be the node that maximises g(i,π_i ∪ {z}) where
             z ∈ Q , z ∉ π_i
7            P_new = g(i,π_i ∪ {z})
8            If Legal (P_new) AND P_new > P_old Then
9                P_old = P_new
10               π_i = π_i ∪ {z}
11           Else
12               Proceed = False
13           End If
14       End While
15   End For
O/P  Set of parents π_i for each of the N variables
```

Algorithm 3.5. The K2 / K3 Algorithm

where $g(i,\pi_i)$ is calculated using either 3.6 or the summation of 3.7 and 3.8 below:

$$\log \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(F_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} F_{ijk} \tag{3.6}$$

$$\log(n) \times \left((r_i - 1) \prod_{j \in \pi_i} r_j \right) / 2 \tag{3.7}$$

$$\sum_{j=1}^{q_i} \sum_{k=1}^{r_i} - F_{ijk} \times \log \left(\frac{F_{ijk}}{F_{ij}} \right) \tag{3.8}$$

The $g(i, \pi_i)$ is used in all of the following algorithms to score networks. However, rather than calculate this for all nodes, it was only applied to nodes at time t. These are the only nodes which have changing parents and so all other nodes scores will remain fixed. What is more, we can calculate these values independently for each variable at time t. This is due to the assumption of no missing data where the score will decompose to the summation of DLs or product of likelihoods [Cooper92].

Genetic Algorithm (adapted for DBN from [Larranaga96])

The Genetic Algorithm searches for the global optimum through the application of recombination and mutation operators as explained in section 3.1. These operators are applied to a population of candidate solutions which we will represent using the triple list method, proposed above, as opposed to the standard binary chromosome. Many different forms of operator exist, the most common being single point crossover and standard mutation. These operators have been adapted for the application to two triple list parents where each parent can be of varying length.

Crossover :

```
I/P  Two triple lists: Par1, Par2 containing len1 and len2
     triples respectively
1    cp1 and cp2 are set to random values in the distribution
     U(0,len1) and U(0,len2), respectively
2    Set Child1 to the triples: { Par1(1,..., cp1) ∪ Par2(cp2,..., len2) }
3    Set Child2 to the triples: { Par2(1,..., cp2) ∪ Par1(cp1,..., len1) }
O/P  Child1, Child2
```

The crossover operator is applied to the triple lists in the same way as it would be to a binary chromosome (see Figure 3.13). Note that the triple lists can vary in length (determining the size of *Q*).

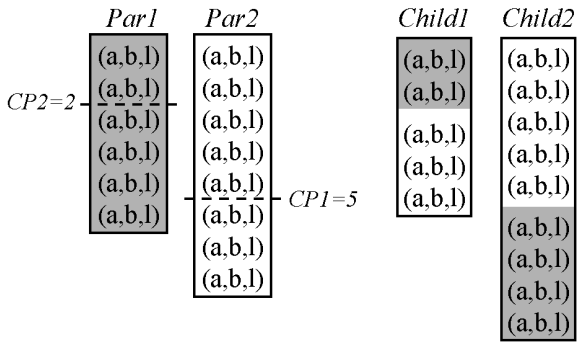


Figure 3.13. The Crossover Operation Applied to two Parent Triple Lists of length 6 and 8 respectively. This operator generates two new Children Triple Lists. Crossover points were 2 and 5 for *Par1* and *Par2* respectively.

Mutation :

The mutation operator involves randomly adding or removing a triple from the triple list in question. Any new triple is generated using random values from uniform distributions of the form:

$$\left(U(0, N-1), U(0, N-1), U(1, MaxLag)\right)$$

Unlike, [Larranaga96] operators on static BNs, these are closed operators. They cannot generate a cyclic graph due to the contemporaneous link assumption within the representation.

The Genetic Algorithm for DBNs is very similar to the standard GA described earlier. It determines parents based on their fitness, the fitter being more likely to be selected. Crossover Rate determines the number of times two parents are selected to perform crossover. Mutation Rate determines the likelihood that a chromosome has one triple mutated.

```
I/P  A (n×N MTS), Crossover Rate, MutationRate, Popsize,
     MaxBranch
1    Initialise random Population of Legal varying length
     triple lists
2    For i=1 to Generations
3        Select Parents from Population based on their
         fitness according to CrossoverRate
4        Generate Children from selected parents using
         Crossover
5        Mutate individuals using Mutation according to
         MutationRate
6        Add all Legal Children to Population
7        Remove the least fittest individuals until
         Population is of size Popsize
8    End For
O/P  The DBN represented by the fittest individual in
     Population
```

Algorithm 3.6. The Genetic Algorithm for Learning DBNs

Evolutionary Program (adapted for DBN from [Wong99])

The Evolutionary Program described here is a simplified version of Wong's EP which was applied to static BNs in that it makes use of a specialised operator called the Knowledge

Guided Operator. This requires calculating the DL of all possible single links in the network in order to bias the mutations. For DBNs the DL must be calculated over all possible time lags as well as between all possible variables. The likelihood of a single link was used to bias the mutation where the log likelihood metric was used.

```
I/P  A (n×N MTS), Popsize, MaxBranch
1    Initialise random Population
2    For i=1 to Generations
3        Generate a child for each member of Population using
         KGM
4        For p=1 to Popsize
5            Select individual p from Population and q other
             random individuals
6            For each random individual with a fitness less
             than the fitness of  individual p add one to its
             score
7        End For
8        Randomly Mutate all individuals in new Population
         and remove any that are not Legal
9        Select the Popsize individuals with the highest
         scores to recreate the next Population
10   End For
O/P  The DBN represented by the fittest individual in
     Population
```

Algorithm 3.7. The Evolutionary Algorithm for Learning DBNs.

Knowledge Guided Mutation (*KGM*) takes a list of all possible links (triples) in the DBN which have been scored according to Equation 3.6, or 3.7 and 3.8. Given a parent, it then randomly adds or deletes a triple where a triple is more likely to be added, the better the its score and more likely to be deleted, the worse its score. Mutation is identical to the Mutation operator applied to the GA.

Branch & Bound (adapted for DBN from [Suzuki96])

The ordering on the nodes can also be ignored for this recursive algorithm. It is more efficient than carrying out an exhaustive search yet is guaranteed to find the optimal structure given a dataset D. It does this by calculating bounds on further recursive calls on a branch. If these bounds are not lower than the current DL then further searching is ignored. This offers an intelligent method of limiting the search, where we normally have to apply a blanket limit on the branching factor of our networks.

```
    Branch_Bound (π1, p1, DL1, E1)
1   Calculate entropy E1 for π1 using Equation 3.3;
    DL1=E1+ p1 ; H1=π1
2   If π1= null Then
3       j = 0
4   Else
5       j = last element in π1
6   End If
7   For q = Q_j to Q_|Q|
8       π2 = π1∪q ;  p2 = p1×r_q
9       If (E1 > p1×(r_q −1)) & (π2 < MaxBranch ) Branch_Bound
        (π2, p2 ,DL2,H2)
        If DL1 < DL2 Then
            DL1 = DL2
            H1 = H2
10      End If
11  End For
```

Algorithm 3.8. The Branch and Bound Algorithm for Learning DBNs.

The recursive function, above, is applied to each of the *N* variables, where $\pi 1$ is initially set to a null list and $p1$ to $\log(n)\times\frac{(r_i-1)}{2}$.

3.5.3 Experimental Results

We found that the best performance over all datasets was achieved using the parameters shown in Table 3.1 for each of the algorithms. Interestingly, the initial population for EP was found to be optimal when all chromosomes contained no triples. This is most likely due to its strength being through the use of pre-processed knowledge as opposed to the recombination of random triples as used by the GA. *MaxBranch* was set to three for all experiments except Branch and Bound in order to limit the search to some degree. Branch and Bound limits the search, itself but a limit of five was still applied as anything greater than this would have seriously affected inference.

	K2 / K3	Branch and Bound	EP	GA
Popsize	1	1	10	100
Crossover Rate	-	-	-	0.8
Mutation Rate	-	-	0.8	0.1
MaxBranch	3	5	3	3

Table 3.1. Parameters for the Adapted Search Algorithms.

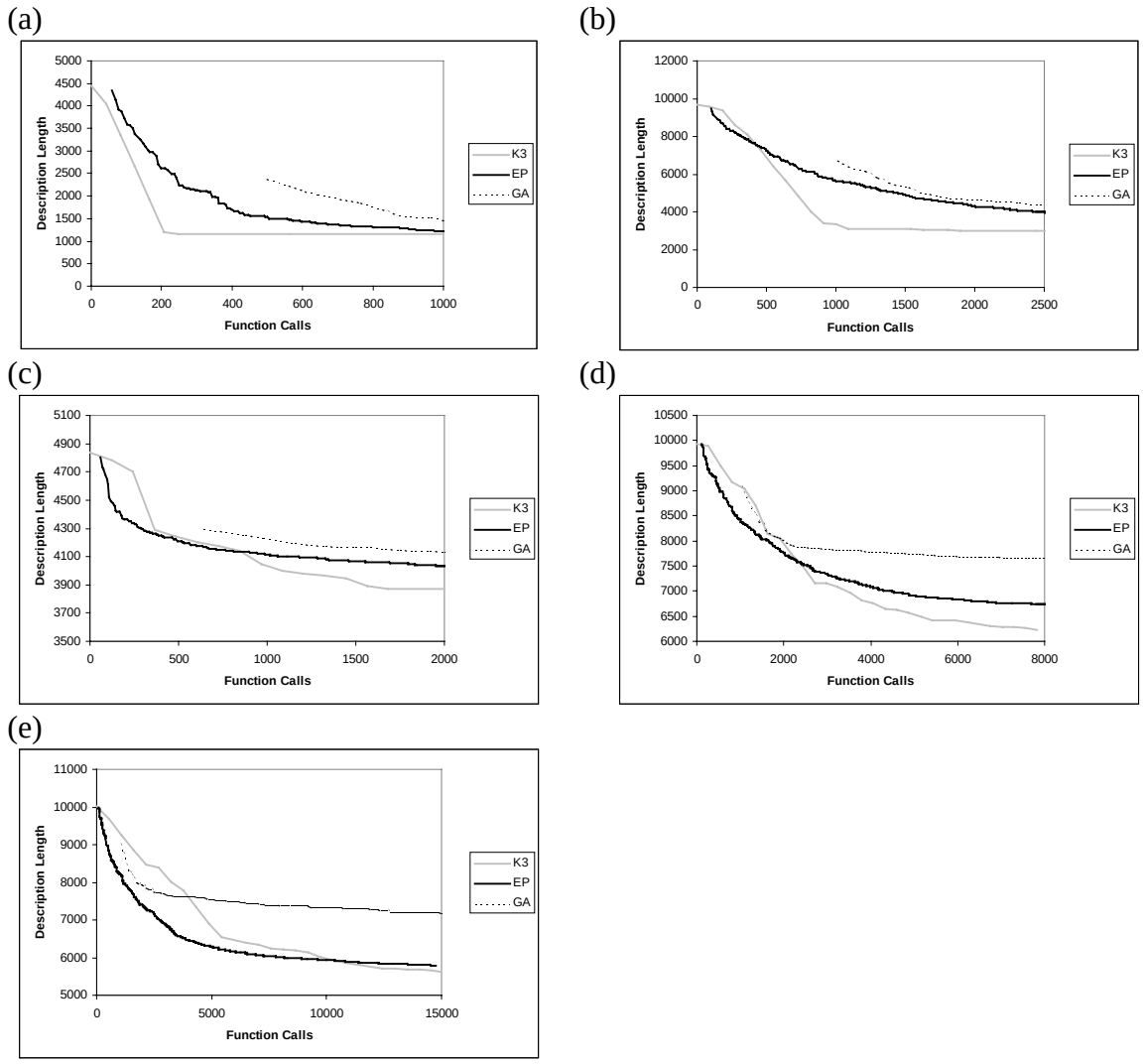


Figure 3.14. Comparing the Search Methods on DBN-Generated MTS using Minimum Description Length (a) N=5, $MaxT$=10; (b) N=10, $MaxT$ =10; (c) N =5, $MaxT$ =30; (d) N =10, $MaxT$ =30; (e) N =10, $MaxT$ =60

It is evident from graphs 3.14(a) and 3.15(a) that on the smaller synthetic datasets the K2 and K3 algorithms are the fastest at finding a good structure. However, these algorithms can suffer from local maxima and some of the experiments using other global methods have found structures with better scores after a larger number of function calls.

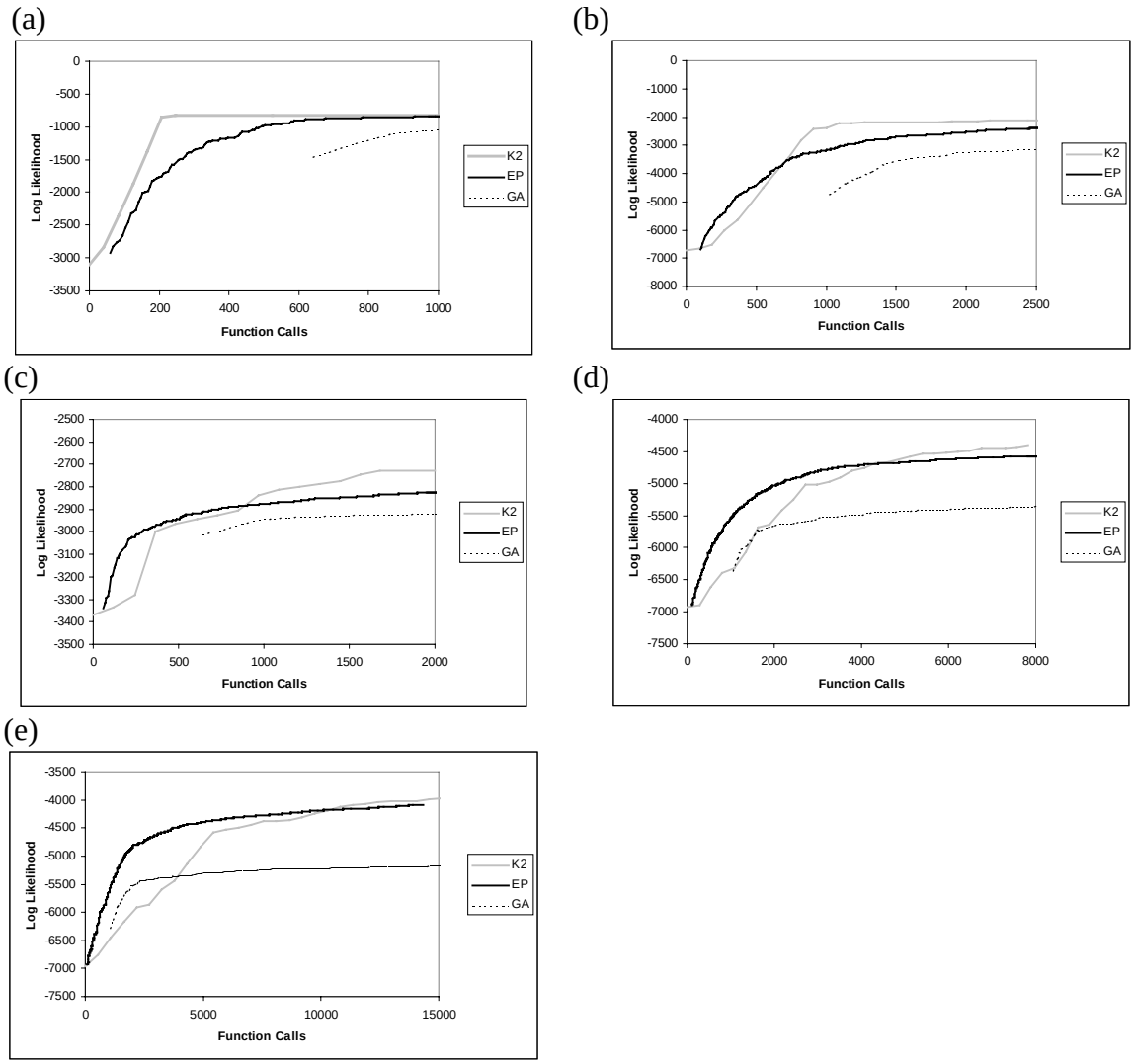


Figure 3.15. Comparing the Search Methods on DBN-Generated MTS using Maximum Log Likelihood (a) *N*=5, *MaxT*=10; (b) *N*=10, *MaxT* =10; (c) *N* =5, *MaxT* =30; (d) *N* =10, *MaxT* =30; (e) *N* =10, *MaxT* =60.

Notice that as either *N* or *MaxT* increases, graphs 3.14(b), (c) and (d), and 3.15(b), (c) and (d), the EP method appears to find better networks in a shorter number of function calls. The KGM heuristic is of assistance in speeding the convergence of the algorithm. What is more, as the networks increase in both dimensionality and time lag, K2 and K3 become less and less efficient. This is probably due to the unnecessary search over the addition of every

possible single link to the network (including all variables and time lags). The GA does not appear to perform that well, particularly in the smaller networks but performs better than K2 and K3 in the earlier generations on larger networks, graphs 3.14(e) and 3.15(e). This is likely to be due to the efficient recombination of good links in the first few generations followed by the reliance upon mutation to fine-tune the DBN further.

Table 3.2 shows us the number of function calls to find optimal structure using Branch and Bound (only applicable to the MDL metric) Branch and Bound is generally the best method for finding the optimal structure in that it is far more efficient than an exhaustive search. However, in considering the problem of finding a good but not optimal structure quickly it is not really a suitable method, requiring many more function calls to find a network that scores comparably to the other search results. Note that we have limited the branching factor (i.e. the number of parents of any node). If it is desirable to perform rapid inference and to acquire networks quickly, it makes sense to control the limit on the branching factor of networks in order to achieve this. Branch and Bound offers a more intelligent method of doing this by ensuring networks of higher connectivity with lower DL are not searched but even on the Branch and Bound method, when N and $MaxT$ are large, the search can become so large as to be unfeasible.

N	*MaxT*	Search Space	Function Calls	MDL
5	10	1.185×10^{7}	1.040×10^{5}	832.665145
10	10	7.938×10^{8}	1.667×10^{6}	2597.160
5	30	3.062×10^{9}	N/A	N/A
10	30	1.992×10^{11}	N/A	N/A
10	60	6.426×10^{12}	N/A	N/A

Table 3.2. The Number of Function Calls to Find *Optimal* Structure using Branch and Bound with MDL. *MaxBranch* has been set to 5. N/A shows where the number of function calls was too large to practically carry out the experiment.

3.6 Conclusions

Firstly within this chapter, issues concerning BNs and DBNs have been documented including learning models from data, inference within models and modelling hidden variables. Next, some preliminary experiments have been carried out to look at the DBN paradigm as a method for automatically generating explanations from MTS. There are various issues that are raised from the experiments such as 'How should continuous data be discretised?' and 'What method should be used for learning the models?'. If these issues can be resolved, it appears that the DBN will offer a way of integrating historical data with transparent and interactive models for querying relationships within data. Finally in this chapter, the issue of learning such models as quickly as possible are looked at through the adaptation of various algorithms for static BNs to learning DBNs, and their efficiency has been compared on synthetic data. If the rapid on-line generation of a DBN is the goal then K2, K3 and Branch and Bound are unlikely to be of use. As *N* and *MaxT* increase, these methods become more and more inefficient. In Chapter 5, evolutionary methods are looked at to quickly learn DBNs by combining the salient features found in the search methods explored in this chapter with the addition of new heuristics based on analysis of process data.

However, in applications where time is not as limited, such as the offline processing of repository data, a more thorough search may be of use. Branch and Bound offers an efficient way to learn models. However, the sheer size of the search space may still make this method unfeasible. To sum up, on datasets with larger dimensionality and larger maximum time lag, an approximate evolutionary algorithm appears to be the most likely candidate for learning a good DBN structure quickly from process data.

4 Grouping High Dimensional Time Series Variables

In this and the next chapter, methods are investigated to try and learn a model for explanation from MTS in as little time as possible in order to make analyses and decisions based on new sets of data. As the dimensionality of MTS and the number of possible time lags increase, the search spaces involved will swell dramatically making real time learning impossible. Therefore, it will be valuable to discover a method that will quickly break down a large dimensional MTS into a number of smaller, relatively independent MTS. In this chapter, methods to *group* variables in MTS are discussed in order to achieve this. Some approximate search methods can then be applied to these subsets of data in order to explain events rapidly. This chapter is based on the work in [Tucker2000] and [Tucker2001a].

4.1 Time Constraints on Learning High Dimensional DBNs

There are many practical applications involving the partition of a set of objects into a number of mutually exclusive subsets. The objective is to optimise a *metric* defined over the set of all valid subsets, and the term *grouping* has been often used to refer to this type of problems. Examples of the grouping applications include bin packing, workshop layout design, and graph colouring [Falkenauer98]. Much research has been done on the grouping problem in different fields, and it was established that many, if not all grouping problems, are NP-hard [Garey79]. Therefore, any algorithm that is guaranteed to find the global optimum will run in exponential time to the size of problem space, and a heuristic or approximate procedure is normally required to cope with most of the real world problems. A variety of techniques have been proposed to develop this procedure, including traditional clustering algorithms, hill-

climbing and evolutionary algorithms. These techniques utilise a *metric* that takes relationships or dependencies between objects into account, and partition them into a number of mutually exclusive subsets [Falkenauer98].

When it comes to the problem of decomposing a high-dimensional multivariate time series (MTS) into a number of low dimensional MTS, the number of possible dependencies between time series variables becomes huge because one variable could affect another after a certain lag. Therefore how to effectively utilise these dependencies becomes an important issue: to use all the possible dependencies in a variable grouping algorithm will be computationally infeasible for many, especially real-time, applications.

This chapter is about a systematic study of the "variable groupings" problem in multivariate time series (MTS). In particular, different heuristic methods are investigated for utilising the information regarding dependencies among MTS variables. In all, 15 such methods are suggested and applied to six datasets where there are identifiable mixed groupings of MTS variables. Finally the most efficient method is applied to oil refinery data and comments from control engineers are discussed.

4.2 Pre-processing MTS using Pair-wise Dependencies

Given a multivariate time series, the aim is to partition the variables into a number of smaller dimensional time series. The methodology introduced in this chapter consists of two stages (see Figure 4.1):

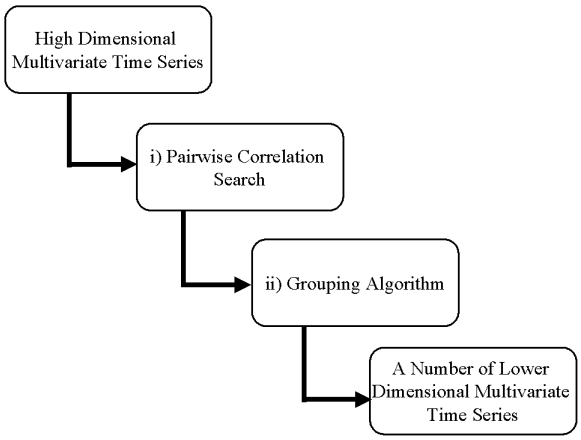


Figure 4.1 - A Process Diagram of the Grouping Procedure

i) Firstly a search over combinations of variables and time lags is carried out in order to find a list of highly correlated variables. Let us call this collection of dependencies *List*, which will be of length *R*. *List* will consist of *triples* where a triple is made up of two variables and a time lag. For example, the triple $(a_i, a_j, 5)$ represents the correlation between a_i and a_j with a time lag of 5. Essentially all of the triples in *List* represent the variable pairs that are deemed to be significantly correlated with the corresponding time lag. Therefore, it is important to estimate what *R* should be with a high degree of accuracy. This is discussed further in section 4.4.

ii) Stage two consists of a grouping algorithm which is applied to *List* where a specifically designed metric is used to group the variables in the original MTS based on the pairs of variables found in *List*. Note that the lag portion of the triple is no longer used once the grouping algorithm is applied. This is because highly correlated variables are to be grouped irrespective of the time lag between them.

This section is arranged as follows. After, outlining the basic notation in section 4.2.1, three methods for generating *List* are introduced in section 4.2.2. These methods are capable of generating a list of highly correlated variable pairs, which can then be used along with an appropriate metric by a grouping algorithm.

4.2.1 Preliminaries

Given a multivariate time series, *A*, with *N* variables and of length *n* the aim is to partition each variable into *m* groups. The set of groups will be denoted by *G* and the size of each group, g_i will be denoted by k_i. This will be achieved by generating a list of 'strong' correlations, *List*, which will be of length *R*. *List* will be calculated by using different searches through the number of possible correlations, *s*, where the number of calls to the correlation coefficient will be denoted by *c*. The aim of this search is to find the true underlying dependencies that generated the data. The number of *explicit* dependencies (or true underlying dependencies) will be denoted by *e*. For a full list of notation used in this chapter, see the glossary in Appendix A.

4.2.2 The Correlation Search

The first stage of the methodology involves searching for the *List* which is the top *R* correlated variables over all possible time lags up to some maximum, *MaxT*. Three methods are discussed for performing this task and these make use of the triple representation described in the previous chapter. The Correlation lists generated using these methods will then be used in conjunction with five different grouping strategies described in section 4.3.

Note that at time lag zero, the correlations represented by the triples *(a_i, a_j, 0)* and *(a_j, a_i, 0)* are effectively the same so duplicates are considered *invalid*. The triples *(a_i, a_i, 0)* are all one, and hence these are considered invalid too. All triples of the form *(a_i, a_i, lag)* will also be considered invalid since these are auto-correlations and do not show relationships between different variables. All invalid triples were removed in all three methods. This results in a search space determined calculated by $N(N-1)(MaxT+0.5)$ given the outlined invalid triples.

The Exhaustive Search

The exhaustive search consisted of simply exploring all of the variables, at each time lag. The algorithm is detailed below.

```
I/P  A (n×N MTS)
1    Set List = Empty List
2    For i = 0 to N-1
3        For j= 0 to N-1
4            For lag = 0 to MaxT
5                If the triple (a_i,a_j,lag) is valid
6                    insert new triple (a_i,a_j,lag) into List
7                    Sort List in descending order of
                     correlation calculated from the MTS
8                    If size of List = R+1 then remove a
                     triple from the tail of List
9                End If
10           End For
11       End For
12   End For
O/P  List of length R
```

Algorithm 4.1 - The Exhaustive Search for *List*

The Random Bag

This is a heuristic approach whereby a random selection of triples is placed in a "bag" containing *R* triples. With each iteration a new random triple is added to the bag. When the

bag overflows, the worst correlation falls out. This is repeated for a predefined number of iterations. The algorithm is described below:

```
I/P  A ( n×N MTS)
1    Set List = Empty List
2    Repeat c times Do
3        i = U(0,n-1), j = U(0,n-1), lag = U(0,MaxT) where
         (i, j,lag) is valid
4        Generate new triple: (a_i, a_j, lag)
5        If triple∉ List then insert into List
6        Sort List in descending order of correlation
         calculated from the MTS, A
7        If size of List = R+1 then remove a triple from the
         tail of List
8    End Loop
O/P  List of length R
```

Algorithm 4.2 - 'Random Bag', A Heuristic Search for Finding *List*

Note that *c* is the maximum number of allowed calls to the correlation function and U(min,max) returns a uniformly distributed random integer between min and max inclusive.

Evolutionary Programming

Evolutionary Programming was introduced in Chapter 3 in the context of learning DBN structure. It is an evolutionary algorithm with the emphasis on mutation and the method does not use any recombination. The basic algorithm employed to discover good triples is outlined as in Algorithm 4.3.

```
I/P  A (n×N MTS)
1    Set List = Empty List
2    Generate R random triples, (a_i,a_j,lag), and insert into
     List
3    Set CallCount = R
4    While CallCount < c
5    Set Children to List
6              Apply Mutate operator to Children
7         Insert valid Children into List
8         Update CallCount by the number of valid Children
9              Apply Survival operator to List
10   End While
O/P  List of length R
```

Algorithm 4.3 - Evolutionary Program for Generating *List*

A Child will be considered invalid if it is already in *List*. Traditionally, EP algorithms use *Tournament Selection* during the survival of the fittest stage (as in the DBN search of Chapter 3) and the best chromosome out of the final population will be the solution to the problem. However, it was decided that the entire population would be the solution for our EP method as in the RB method. That is, each individual chromosome would represent a single correlation (a triple) while the population would represent the set of correlations found (*PopulationSize*=*R*). Hence the survival operator consisted of keeping the best *R* individuals. This therefore required a check for any duplicates after mutation, and for any invalid chromosomes. Any children that fell into this category were repeatedly mutated until they became valid. Although the entire population would represent the solution, it must be noted that the fitness of each individual would still be independent of the rest of the population. Each individual would try to maximise the correlation coefficient that it represents. This in turn would maximise the population's fitness by improving the set of correlations represented by the population.

Within the EP, a gene is either a variable, a_i or the *lag*. The idea of *Self-Adapting Parameters* [Baeck96] has been used in this context to mutate the genes using a normal distribution that is rounded up to the nearest discrete value. Whilst this is unlikely to have any effect on mutating variables (the ordering of the variables is arbitrary), it is hoped that this controlled mutation will assist the EP in 'homing in' on the best time lag. Here each gene, $gene_i$, in each chromosome is given a parameter, σ_i. Mutation is defined as follows:

$$gene_i = gene_i + N(0, \sigma_i) \quad (4.1)$$

$$\sigma_i = \sigma_i \cdot \exp(N(0, \tau) + N(0, \tau_i)) \quad (4.2)$$

$$\tau = \frac{1}{\sqrt{2len}} \quad (4.3)$$

$$\tau_i = \frac{1}{\sqrt{2\sqrt{len}}} \quad (4.4)$$

Note that τ is constant for each gene in each chromosome but different between chromosomes, and τ_i is different for all genes. Both parameters are generated each time mutation occurs. Each chromosome consisted of three parameters and their corresponding σ_i values. The value of *len* is the size of each chromosome, i.e. three. Each gene within a chromosome is mutated according to the Normal distribution with mean 0 and standard deviation equal to the gene's corresponding standard deviation, σ_i, in Equation 4.1. Each σ_i is then mutated according to Equation 4.2.

4.3 The Grouping Algorithms and Metric

In this section the partition metric is introduced and its properties discussed in 4.3.1. This is then followed by the description of the different search algorithms in 4.3.2.

4.3.1 The Partition Metric

The Partition metric, defined below, is used to group variables together where they have strong mutual dependency and to separate them into different groups where the dependency is low. Let N be the number of variables, G be the list of groups and $m = |G|$ (the number of groups). Let g_i be the ith member of the list G where $1 \leq i \leq m$ and let $k_i = |g_i|$. The notation g_{ij} refers to the jth element of the ith set of G. It is clear that in all cases $m \leq N$. The *partition metric* for any fixed list G, $f(G)$, is defined as follows, where $corr(x_i, x_j)$ returns true if there exists in *List* any triple of the form (x_i, x_j, lag) or (x_j, x_i, lag) for any valid *lag*.

$$f(G) = \sum_{i=1}^{m} h(g_i) \tag{4.5}$$

$$L(g_{ia}, g_{ib}) = \begin{cases} 1 \text{ if } \text{corr}(g_{ia}, g_{ib}) \\ 0 \text{ if a} = \text{b} \\ -1 \text{ otherwise} \end{cases} \tag{4.6}$$

$$h(g_i) = \begin{cases} \sum_{a=1}^{k_i} \sum_{b=1}^{k_i} L(g_{ia}, g_{ib}) \text{ if } k_i > 1 \\ 0 \qquad\qquad\qquad \text{otherwise} \end{cases} \tag{4.7}$$

The metric has the following characteristics (proofs for these can be found in Appendix B):

1. If there are no correlations, the maximum value is obtained when all variables are in separate groups.
2. If a correlation exists for each pairing of variables (the search space), then the maximum fitness is obtained when all of the variables are in one group.
3. If the data generating the correlations came from a mixed set of MTS observations, then the metric will be maximised when the variables within the same group have as many correlations within the list Q as possible and variables within differing groups contain as few correlations as possible.

In the following experiments a *correl* has been chosen that is a well established correlation coefficient - Spearman's Rank Correlation [Snedecor67]. Spearman's Rank Correlation (SRC) measures linear and non-linear relationships between two variables, either discrete or continuous, by assigning a rank to each observation. The SRC can be calculated between two variables over differing time lags by shifting one variable in time. The equation incorporates the sums of the squares of the differences in paired ranks, according to the formula:

$$correl(a_i, a_j, lag) = 1 - \left(\frac{6 \sum_{p=1}^{n-lag} \left(rank(a_i(t)) - rank(a_j(t+lag)) \right)^2}{(n-lag)((n-lag)^2 - 1)} \right) \quad (4.8)$$

where n is the length of the MTS and $rank(a_i(t))$ is calculated from ordering and ranking every observation of the variable a_i on its value and recording the rank of the value at position t.

It should be noted that the methods are in no way restricted to using this particular coefficient and others such as Pearson's could have easily been used. Spearman's Rank was chosen as it is well recognised and not restricted to finding linear dependencies.

4.3.2 The Grouping Search Algorithms

Various methods have been investigated for maximising the partition metric outlined above in the context of grouping MTS. First of all the adopted Genetic Algorithm approach is described followed by three different forms of this algorithm. Next a hill climb technique is described and finally a heuristic clustering method.

The Genetic Algorithms

The general Genetic Algorithm [Holland95, Goldberg89] described below uses the notion of *chromosomes* which represent possible solutions to a particular problem. *Crossover* and *Mutation* operators are applied to these chromosomes in order to search different possible solutions and a selection process is applied to the *population* of chromosomes in order to preserve 'good' solutions and discard 'poor' ones. Our general algorithm for generating a set of groups, *G*, from a set of correlations, *List*, is given in Algorithm 4.4.

```
I/P  List
1    Generate Population chromosomes and calculate their
     fitness according to Equation 4.6
2    For i = 1 to Generations do
3        For j = 1 to CrossoverRate × Population do
4            Set Parent1 to a random chromosome (with fitter
             chromosomes being chosen with higher
             probability)
5            Set Parent2 to a different random chromosome
             (with fitter being chosen with higher
             probability)
6            Apply Crossover Operator to Parent1 and Parent2
             to generate Offspring1 and Offspring2
7            Apply Mutation Operator to Offspring1 and
             Offspring2
8            Insert Offspring1 and Offspring2 into the
             population
9            Sort the population according to Fitness
10           Remove the chromosomes with the least Fitness
             but retain the Population fittest chromosomes
11       End For
12   End For
O/P  G (a set of groups, constructed from the fittest
     individual from the final population)
```

Algorithm 4.4 - Standard Grouping GA

The following describes three different representations, forms of crossover and mutation that were used with this general algorithm. For the scope of this chapter, the fitness function for the methods will be the partition metric defined in Equation 4.6.

1) Gene Per Variable (GPV)

This representation consists of a chromosome with each gene representing a variable in the domain. The value of the gene determines which group the variable is a member of. For example, 10 variables being placed into 3 groups:

Group 0: 0 3 8
Group 1: 2 7 4 1 5 } would be represented by the following chromosome: 0 1 1 0 1 1 2 1 0 2
Group 2: 6 9

The Crossover operator used for this representation is Holland's [Holland95] standard one point crossover and the Mutation operator involves randomly mutating genes within the chromosome according to the Mutation Rate. Each gene has Mutation Rate probability of being mutated to a value from a uniform distribution U(0,*n*-1).

2) Goldberg's Partially Mapped Crossover (PMX)

This form of crossover applies to a new representation of the grouping problem where the chromosome consists of variables interspersed with group dividers. For example, let a group divider be represented by the symbol $\square_i$ where the subscript is unique and each of 10 variables within a domain be represented by a unique integer.

Therefore the chromosome: 0 3 8 $\square_1$ 2 7 4 1 5 $\square_2$ 6 9 would represent the groupings in the previous example. In other words, variables within the same group dividers will be classed within the same group.

This representation requires a new crossover operator in order to ensure that invalid offspring are not produced. It can be seen that standard crossover as used in the GPV representation would produce many invalid offspring as it would be highly likely to result in offspring with variables appearing in more than one group. Goldberg introduced the PMX operator [Goldberg85] which prevented this and developed an o-schema theory (closely linked to Holland's original schema theory). It ensures all offspring are valid (i.e. it is a closed operator) and works as follows:

i) Select two crossing points for both parents

ii) Swap all elements between the crossing points

iii) For all repeating elements in the old part of the chromosome, replace with the value found on the corresponding position on the other chromosome.

For example:

Parent 1: 4 $\square_3$ 0 $\square_1$ 1 6 5 2 $\square_2$ 3 & Parent 2: 5 4 $\square_2$ 2 3 $\square_3$ 0 1 $\square_1$ 6

i) Crossing points = 3 and 5

ii) Swap elements "$\square_1$ 1 6" with "2 3 $\square_3$" 4 $\square_3$ 0 2 3 $\square_3$ 5 2 $\square_2$ 3

5 4 $\square_2$ $\square_1$ 1 6 0 1 $\square_1$ 6

iii) Replace repeated values 4 6 0 2 3 $\square_3$ 5 $\square_1$ $\square_2$ 1

5 4 $\square_2$ $\square_1$ 1 6 0 3 2 $\square_3$

Mutation involves randomly mutating genes within the chromosome according to the Mutation Rate. Each gene has Mutation Rate probability of being mutated to a value from a uniform distribution U(0,2*n*-1), where values greater than *N*-1 were replaced by dividers with unique indices.

3) Falkenauer's Grouping Genetic Algorithm (GGA)

This representation is similar to the GPV except that it also has an extra part on the chromosome which represents the actual groups without any information about their contents. For example the same groupings as the previous examples would be represented by the following chromosome: 0 1 1 0 1 1 2 1 0 2 : 0 1 2

It is the second part of the chromosome (after the colon) that crossover is applied to. Crossover works as follows:

i) Select two random crossing sites, delimiting the crossing section in each of the two parents.

ii) Inject contents of the crossing section of second parent at the first crossing site of first parent.

iii) Remove any elements that are repeated from the groups that were members of in the first parent.

iv) Remove any empty groups and reinsert any unassigned variables to existing groups.

v) Repeat (i) to (iv) to produce the second offspring by reversing the roles of the first and second parent.

Example for first offspring:

Parent 1: 0 1 1 0 0 2 1 2 : 0 1 2 & Parent 2: 4 5 3 4 5 6 3 6 : 3 4 5 6

i) Cross Sites: Parent 1 = [0,1], Parent 2 = [1,3]

ii) Inject group 0 into position 1 0 ? ? 0 0 ? ? ? : 3 0 4 5 6

iii) Remove group 4 and 5 due to repeats 0 ? 3 0 0 6 3 6 : 3 0 6

iv) Reinsert variable 1 into random group (6) 0 6 3 0 0 6 3 6 : 3 0 6

where ? denotes an unallocated variable (adapted from [Falkenauer98]).

Mutation involves randomly mutating genes within the chromosome according to the Mutation Rate. Each gene has Mutation Rate probability of being mutated to a value from a uniform distribution U(0,*n*-1). This is identical to the mutation used within GPV. However, it is only applied to the first part of the chromosome and the second part, after the colon, is updated accordingly.

Falkenauer proves [Falkenauer98] that this method allows the schema theory to hold even for grouping problems. In contrast, PMX and standard crossover as used in GPV, with their schema and o-schema theories, appear to collapse when applied to these sort of problems.

Hill Climbing

A Hill Climbing Search is essentially an iterative procedure that continually moves in the direction of increasing value for some metric. Our version of Hill Climb involves using the GPV representation and making simple changes to the current groupings with each iteration. Within each iteration one variable is moved into another existing group or placed into a

newly formed group and if this change improves the score of the individual, it is retained. The algorithm is outlined below.

```
I/P  List
1    Generate a random selection of groupings (i.e. a single
     chromosome using the GPV representation)
2    Set Score according to the Partition Metric applied to
     List given the grouping
3    For i = 1 to Iterations do
4         Make a random change to the groupings (either merge
     two groups or split a group into two)
5         Set New_Score according to the Partition Metric
     applied to List given the new grouping
6         If New_Score < Score Then undo changes
7    End For
O/P  G (a set of groups)
```

Algorithm 4.5 - Grouping Hill Climb

Mirkin's Separate and Conquer

This method is based on the clustering technique of Separate and Conquer [Mirkin99]. The algorithm had to be amended to allow it to cluster on the relationships between variables rather than on the value of variables. The algorithm is as shown in Algorithm 4.6 and uses Equation 4.6 to calculate $h(g_i)$.

To summarise, a new group is created containing the two variables that have the highest correlation between them. The next step is to take each variable in turn, and iterate through each group that exists, seeing if adding the variable to that group increases the groups' score. If this is the case, then the variable is added to that group. If there are no more groups to test a given variable with, then it is placed into a new group on its own.

```
I/P  List
1    Let G be a set of Groups (empty)
2    Let A be a set of variables {1..N}
3    Create a group g1 containing the best correlation pair
     in List
4    Add g1 to G
5    m=1
6    For i = 1 to N
7         Set skip=false
8         j = 1
9         While j < m+1 and skip=false
10             If ai ∉ gj then
11                  Add ai to gj to create g'j
12                  If h(g'j) > h(gj) then
13                       Add ai to gj
14                       skip=true
15                  End If
16             End if
17             j= j+1
18        End While
19        If skip =false then
20             Create a group g* containing only ai
21             Add g* to G
22             m=m+1
23        End If
24   End for
O/P  G (a set of groups)
```

Algorithm 4.6 - Separate and Conquer

4.4 Parameter Estimation

In order to retrieve groupings that correspond closely to the correlations that represent actual dependencies, the ideal set of parameters will have to be determined for the correlation search, most importantly *R*, the size of the *List*. As this will determine the cut off point for significant correlations, it will affect the overall algorithm a great deal. For example, a cut off point that is too high will mean there are too few significant correlations resulting in smaller groups; a cut off point that is too low will mean there are too many significant correlations

and so groups will be combined into larger ones due to the inclusion of low correlated variables in the list. It has been decided to try and determine the parameters through simulations of the random bag method described within this section. Random Bag was chosen since it is the simplest to model. It should also be the weakest of the three methods for correlation search and so by coming up with confidence intervals for selecting all the true correlations for this method should result in a worst case scenario for the chosen parameters; namely 95% confidence on Random Bag should mean *at least* 95% confidence on EP. This has been shown to be true in previous work in [Swift99b], and through the experiments within this chapter. These simulations were used to generate probability distributions of selecting correlations that represent actual dependencies. These distributions could in turn be used to determine confidence limits for *R* and the number of calls to the correlation function.

4.4.1 Simulations of Random Bag

Simulations were carried out in order to mimic the way in which the random bag searches for good correlations. These consisted of setting the size of *List* (*R*), the size of the total search space (*s*) and the number of calls to the correlation function (*c*) to particular plausible instantiations and then simulating the act of randomly selecting a correlation from the search space and then recording whether it was a pre-defined "true" dependency. This process can be compared to repeatedly picking a selection of c random cards from a pack without replacement and recording the number of Aces found. Therefore for this case $R = 4$ (the number of Aces) and $s = 52$ (the number of cards in a pack). It was possible, therefore, to generate approximations of the distributions associated with the probability of picking a "true" dependency. The number of these "true" explicit dependencies will be referred to as *e*,

which is always less than or equal to *R*. These distributions were then tested for normality using the Lilliefors' test (in section 4.4.2). The mean and standard deviation were then calculated for each distribution so that a method for symbolic regression could be used to learn a function to determine the mean and standard deviation given *R*, *s* and *c* (in section 4.4.3). The algorithm for simulation is shown in Algorithm 4.7. This was repeated for N_{sims} different values of *R*, *s* and *c*

```
I/P  R, e, s, c and SimulationSize
1    Set dependencies = e randomly selected correlations
2    Set Distribution to be a zero array of length R
3    For i = 1 to SimulationSize
4         count = 0
5         For j = 1 to c
6              Randomly choose R different correlations
7              If (a_i,a_j) is in dependencies Then count =
               count + 1
8         End For
9         Distribution_count = Distribution_count + 1
10   End For
O/P  Distribution
```

Algorithm 4.7 - Stochastic Simulation of Random Bag

The probability distribution for selecting a true dependency is found by dividing each element in the distribution array by *SimulationSize*. *SimulationSize* is a variable that dictates the number of times the process is repeated to ensure that a good approximation to the random bag process is reached.

4.4.2 Lilliefors' Test

Lilliefors' test [Lilliefors67] is a simple test for normality that can be performed on a known distribution function. The simulations performed in section 4.4.1 can easily be transformed

into the required format for this method and the test can be performed to see if the random bag method can be approximated by a normal distribution. The test is as follows:

Given *v* observations, a metric D_{max} is computed by

$$D_{\max} = MAX\left|F^*(e) - C_v(e)\right| \tag{4.9}$$

Where $C_v(r)$ is the sample cumulative distribution function, $F^*(r)$ is the cumulative normal distribution with μ equal to the sample mean, σ^2 equal to the sample variance, and *v* is equal to *R*+1. Within the simulations, these two summary statistics can be computed directly from the data. If the value of D_{max} exceeds the critical value supplied by Lilliefors in his paper, one rejects the hypothesis that the observations closely follow the normal distribution. For the purpose of this chapter confidence was set at the 99% level, which requires D_{max} not to exceed $\frac{1.031}{\sqrt{v}}$. From the results of these tests it can be assumed that the random bag can be approximated by a normal distribution with a 99% certainty. In fact all of the 150 simulations passed the test for normality at this level. See Appendix C for the full results.

4.4.3 Finding μ and σ

Once it has been ascertained that the distribution of the Random Bag process can be approximated as Normal, a value for the means and standard deviation is needed in order to place confidence limits on the number of function calls needed to find the required *R*, the size of *List*. Since the process itself does not have a very easy representation for the probability distribution, the algebraic representation for the mean and standard deviation is likely to be difficult to derive. Since many simulations have been performed, tabulating *R*, *c*, *s* and the associated μ and σ, these can be used to evaluate the relationship between μ and σ. It is

assumed that μ is a function of *R*, *s* and *c*, and that *σ* is another function of *R*, *s* and *c*. The Genetic Programming technique of Symbolic Regression is used for this [Koza92].

The functions for μ and σ, which shall be denoted $\mu(R,c,s)$ and $\sigma(R,c,s)$, are assumed to be functions in terms of the operators +,-,×,/, and the terminal symbols *R*, *s*, *c* along with the constant integers 0 to 9. The exact form is unknown. A binary tree is used to represent a regular expression in terms of these symbols, with the terminal nodes being a variable or constant and the non-terminals being an operator. The worth of any given tree (its fitness) is the difference between the observed value of μ and/or σ vs. the calculated value, using the equation formed from the tree, and all of the available data. This is defined as in Equation 4.10 and 4.11.

$$Fitness\ \ for\ \mu = -Nodes(\mu) \cdot \sum_{i=1}^{N_{sims}} \left| \mu(R_i, c_i, s_i) - \mu_i \right|^2 \tag{4.10}$$

$$Fitness\ \ for\ \sigma = -Nodes(\sigma) \cdot \sum_{i=1}^{N_{sims}} \left| \sigma(R_i, c_i, s_i) - \sigma_i \right|^2 \tag{4.11}$$

Where *Nodes(●)* represents the number of nodes in the corresponding binary tree, and *i* indexes a variable from the table of simulated examples (where there are a total of N_{sims} examples). As with a Genetic Algorithm, the initial population will be a certain number of random binary trees as described above. This population will be improved (better fitness) over subsequent generations through the use of the standard genetic programming operators of Mutation and Crossover. Note that the negative fitness function ensures that the process tries to improve the population by minimising the fitness. Adjusting the fitness by penalising it on the tree's size will force the genetic program to look for a smaller tree.

Parameters and result statistics of these experiments can be found in Appendix E. The resulting functions for μ and σ were as follows:

$$\mu = \frac{2cR}{2s + c} \tag{4.12}$$

$$\sigma = \frac{R}{63} + \frac{11c}{s} \tag{4.13}$$

4.4.4 Confidence Limits on *c*

Once values for the mean and standard deviation have been found, one can place confidence limits on the probability of the random bag finding a number of correlations that lie between *r* and *R*, where *R* is the size of the random bag and *e* is the number of correlations being searched for. This is the cumulative normal distribution where the probability that the number of correlations found is greater than *r*. For the purpose of this paper, the ratio of *R* to *e* has been chosen as 5 and the confidence limit as 95%. The aim of this exercise is to recommend a value for *c* based on the known parameters *R*, *e* and *s*. Given that the pr(number of correlations ≥ *e*) = 0.95, the standard normal distribution tables can be used with $z = (e\text{-}\mu) / \sigma$ to find what the corresponding value of *c* should be. For the 95% level, the value of *z* should be –1.645. Since μ and σ are known, an equation can be formed in terms of *e*, *R*, *s* and *c* where only *c* is unknown, starting with:

$$z = \frac{e - \mu}{\sigma} \tag{4.14}$$

$$z = \frac{r - \left(\frac{2cR}{2s + c} \right)}{\left(\frac{R}{63} + \frac{11c}{s} \right)} \tag{4.15}$$

Unfortunately this requires a lot of algebra to solve the above equation for *c*. The final solution is a quadratic equation, and when some reasonable approximations are made, is as follows:

$$c \approx \frac{s}{22}\left((1.3e - 2R) + \sqrt{(2R - 1.3e)^2 + \frac{88}{63}(Rz - 63e)}\right) \tag{4.16}$$

The parameter *c* is a guide towards how long the procedure is going to take, in terms of how many correlation function evaluations are made. For example if *c* is greater or equal to the number of calls made by the exhaustive search (*s*), then it is pointless to use the random bag to locate the required number of correlations. As a guideline, a 95% confidence at finding the required number of correlations is aimed for.

4.5 Experiments

The generated synthetic datasets are described in section 4.5.1 and the results of estimating the parameters for the algorithms in section 4.5.2. a metric for evaluating the discovered groupings is then described in section 4.5.3.

4.5.1 Multivariate Time-Series Datasets

Based on the two problems being tackled by the grouping methods - the search for DBN structure and the generation of VAR models, two types of datasets have been produced. One set has been generated by hand-coded DBNs and the other by VAR models. 5 datasets have been generated of each type with varying dimensionality and order. These are described below and the full details for the models used can be found in Appendix D. For the experiments, various variables from these datasets were mixed to produce some which had

only DBN generated data, some which had only VAR generated data and some with a mixture of the two. This was to see how the methods performed under different conditions and for different types of data.

Dataset Organisation

Table 4.1 describes the datasets that were generated using the two methods described above:

	Dynamic Bayesian Network				
MTS	a	b	c	d	e
Order	10	20	5	30	60
Dimensionality	3	5	5	10	10
	Vector AutoRegressive Process				
MTS	f	g	h	i	j
Order	2	3	4	5	2
Dimensionality	10	7	6	3	2

Table 4.1 - The different MTS descriptions.

These 10 multivariate time-series were grouped into various different combinations to produce 6 datasets. The first consisted of all 61 variables, the second consisted of only DBN generated data, the third only VAR generated data and the remaining three consisted of various mixtures. All datasets except the first consisted of 28 variables so as to keep the search space identical. Table 4.2 shows the breakdown of each dataset:

Dataset	1	2	3	4	5	6
MTS	a b c d e f g h i j	a b d e	f g h i j	a e f i j	a b c g h j	d g h i j

Table 4.2 - The Breakdown of each Dataset.

4.5.2 Parameter Estimation Results

If the parameter estimation analysis from section 4.4 is applied to Datasets 1-6, the results obtained are as listed in Table 4.3.

Parameter	Dataset 1	Dataset 2-6
MaxLag	75	75
n (number of variables)	61	28
r	150	64
R	750	320
c	72201	15585
$s = n(n-1)(MaxLag + 0.5)$	276330	57078
$\mu = \frac{2cR}{2s + c}$	173.321	76.879
$\sigma = \frac{R}{63} + \frac{11c}{s}$	14.779	8.083
$z = \frac{r - \mu}{\sigma}$	-1.578	-1.593
$\gamma = c/s$	0.273	0.261
$\beta = r/R$	0.200	0.200
Confidence	0.943	0.945

Table 4.3 - Parameters for Datasets 1-6

The equation for *s* represents the total possible number of correlations at varying time lags, once invalid correlations are removed. μ and σ are defined in Equations 4.12 and 4.13 respectively, *z* is the standard normal variable, and *c* is defined by Equation 4.16. Two new parameters are introduced: γ and β. γ is the ratio of *c* to *s* and gives an indication of how efficient the procedure is going to be. As a guideline, it is recommended that for the random bag to be effective, this value should be less than 1/3. However, the parameter β needs

defining. This represents the ratio *r/R*. a value of 0.2 is suggested, this being found by experimentation, and provides a good trade off between the number of calls to the correlation function, *c*, and how many correlations needed to be stored in memory. As can be seen, the use of the approximation in Equation 4.16 has resulted in the confidence limit not being exactly 95%, but rather 94.4% (on average).

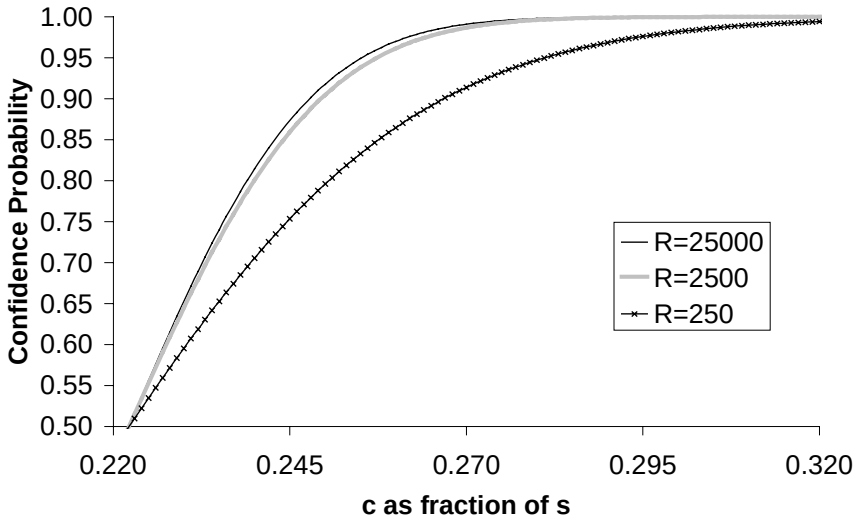


Figure 4.2 – Confidence against γ with varying R

Figure 4.2 shows an example where s = 1,000,000, $\beta = 0.2$, γ is allowed to vary between 0.22 and 0.32, and three values of R are displayed. The values of R corresponds to 2.5%, 0.25% and 0.025% respectively of s. It can be immediately seen from the graph that R=250 requires more correlation function calls for any level of confidence than for the other two values of R. However there is not much between R=25,000 and R=2,500. Other similar experiments have shown that the optimal value for R/s is near the 0.25% mark.

To conclude this section, it has been shown that *c* should be calculated from Equation 4.16 once a confidence limit has been assigned (e.g. 95%, giving a value for *z*: -1.645); here a recommended value for *R/s* is about 0.25%. And finally, it has been found that having the ratio of *r* to *R* being 0.2 proves to be efficient.

4.5.3 Evaluation Metric

A metric is needed to show how similar or dissimilar two groups are so that the results can be compared to the original groups. This is defined by pairing all of the variables up and incrementing the score each time that the pair appears in the correct group within the two groups or when the pair appears in different groups. The metric is scaled so that it returns a value between 0 and 1 inclusive, where 0 represents very dissimilar groups and 1 represents very similar groups. This metric $EVM(G_1,G_2)$ is shown in Algorithm 4.8.

```
I/P  G1 and G2 be two groupings
2    Let n be the number of variables
3    Let EVM = 0
4    For i = 1 to n −1
5         For j = i+1 to n
6              Let g1 be the group within G1 containing i
7              Let g2 be the group within G2 containing j
8              If j in g1 and i in g2 then EVM = EVM + 1
9              If j not in g1 and i not in g2 then
                    EVM = EVM +1
10        Next j
11   Next i
12   Update EVM to 2EVM / n(n−1)
O/P  EVM
```

Algorithm 4.8. The Evaluation Metric $EVM(G_1,G_2)$

Having determined the parameters for the synthetic datasets, the next section presents the results of applying the different grouping algorithms and compares the resulting groups to the original ones using the metric defined in 4.5.3.

4.6 Results from Synthetic Data

This section documents the results of numerous experiments which compare different grouping strategies consisting of all combinations of the proposed methods for grouping search and for correlation search. These 15 strategies are applied to six datasets where there are identifiable mixed groupings of MTS variables. For each experiment the following have been recorded:

i) The Partition metric of the best solution after a varying number of calls to the fitness function for various different datasets. This is a measure of how well the groupings represent the correlations that were discovered during the correlation search.

ii) The score as calculated by the Evaluation metric described in 4.5.3, which is independent of the correlation search results. This can be considered as a measure of accuracy of the resulting groupings. It is essentially a measure of distance between the groups that were used to generate the data and the resultant groups found using our methods.

iii) The number of function calls to find the solution with the highest Partition metric. This can be thought of as a measure of efficiency.

All stochastic grouping algorithms (all methods except Separate and Conquer) were repeated 10 times and the average recorded in order to remove any sampling bias. the marginal

statistics were then calculated over the correlation searches, the grouping strategies and the datasets.

4.6.1 The 15 Methods

In the following tables, the abbreviations *HC* stands for Hill Climbing and *S&C* for Separate and Conquer, and *Ex* for the Exhaustive Search.

	RB/GPV	**RB / PMX**	**RB/ GGA**	**RB / HC**	**RB / S&C**
Partn Metric	110.60	114.90	121.10	125.00	113.67
Eval Metric	0.91	0.92	0.93	0.92	0.90
FC	232292.52	12974.00	8697.67	4881.20	400.67

Table 4.4 - The 5 Grouping Strategies applied to the Random Bag *List*.

	EP/GPV	**EP / PMX**	**EP / GGA**	**EP / HC**	**EP / S&C**
Partn Metric	105.43	109.87	113.67	118.80	109.33
Eval Metric	0.90	0.90	0.92	0.91	0.89
FC	232327.65	10545.67	8152.33	5331.52	427.67

Table 4.5 - The 5 Grouping Strategies applied to the Evolutionary Program *List*.

	Ex/GPV	**Ex / PMX**	**Ex / GGA**	**Ex / HC**	**Ex / S&C**
Partn Metric	117.80	122.50	128.87	130.03	122.67
Eval Metric	0.91	0.92	0.93	0.93	0.92
FC	232237.37	11325.67	8693.67	4778.47	411.67

Table 4.6 - The 5 Grouping Strategies applied to the Exhaustive Search *List*.

It can be seen from the results of the 15 different methods that whilst there is a lot of variation in the number of calls to the Partitioning Function (FC), the metrics, in particular the evaluation metric does not vary a great deal at all. This implies that the initial process of searching for *List* does not have to be exhaustive to get good results. This property would be

very useful for those applications where the partitioning of a MTS must occur on a real time basis. By far the fastest to converge is the Separate and Conquer Method taking little more than 400 function calls. However, it must be noted that this method is deterministic and is not guaranteed to find the best groupings.

The most important statistic is the evaluation metric and the method that seems to perform best over all the datasets is the Exhaustive Search / Hill Climb. Although the GGA finds just as good a solution, it takes almost twice as many function calls. However, as the marginal statistics will show, the GGA method performs better when averaged over all the correlation search strategies. Therefore, it appears that if the exhaustive search cannot be carried out then a combination of Random Bag or Evolutionary Program with GGA is the best option.

4.6.2 A Note Regarding RB and EP

Table 4.7 displays the average of the top *r* correlations for each method of correlation mining, and displays the average over all of the datasets.

Dataset (*r*)	**Exhaustive Search**	**Random Bag**	**Evolutionary Programming**
Dataset 1 (150)	0.592	0.527	0.547
Dataset 2 (64)	0.536	0.488	0.402
Dataset 3 (64)	0.694	0.629	0.659
Dataset 4 (64)	0.641	0.575	0.569
Dataset 5 (64)	0.548	0.509	0.497
Dataset 6 (64)	0.625	0.568	0.558
Dataset's Average	0.606	0.549	0.539

Table 4.7 - The Top *r* Correlations for Three Search Methods.

As can be seen in the table, the exhaustive search performs the best, followed by the random bag, and then evolutionary programming. It should be noted that this result only applies to the situation where there are a large number of correlation calls made (*c* is large). It has been shown in [Swift99b] that the EP method outperforms the RB method for smaller values of *c*. Based on the extensive analysis and experiments performed so far it is recommended that if *c* is more than 30% of *s* then the exhaustive search method should be used. If this is not the case and if *z* corresponds to less than 50%, use the EP method, otherwise use the RB method.

4.6.3 Marginal Statistics

In order to explore more fully the effect of the different correlation searches, grouping strategies and datasets, various marginal statistics have been calculated. Essentially this involved averaging over the correlation searches, the grouping strategies and the datasets to see how each of these methods compared. These results can be found in Tables 4.8 to 4.10 below.

	Average Exh	**Average EP**	**Average RB**
Partition Metric	124.373	111.420	117.053
Evaluation Metric	0.923	0.905	0.915
Function Calls	51489.367	51356.967	51849.210

Table 4.8 - Averaging over Correlation Search

The correlation summary statistics support the conclusion that the method used for generating a good set of correlations does not have a very significant effect on the final groupings. In other words, the evaluation metric which measures the distance between the original groupings and the discovered groupings are very similar for all correlation search methods

(approximately 0.9). Therefore, it would make more sense to perform a fast approximate correlation search on datasets where the search space is so large that the exhaustive search is infeasible.

	Average GPV	**Average PMX**	**Average GGA**	**Average HC**	**Average S&C**
Partition Metric	111.278	115.756	121.211	124.611	115.222
Evaluation Metric	0.907	0.9156	0.926	0.922	0.903
Function Calls	232285.844	11615.111	8514.556	4997.061	413.333

Table 4.9 - Averaging over Grouping Strategy.

The best grouping strategies, as shown by the grouping summary statistics, are the Hill Climb method and Falkenauer's GGA. This is probably due to the economical use of function calls made by Hill Climb (unlike the GA methods which require evaluating populations) and the efficient crossover developed by Falkenauer. The other GA methods used less efficient crossovers and the Separate and Conquer method is deterministic and therefore can never be guaranteed to find the global solution. It is, however, very fast at finding a good set of groupings after a very small number of function calls.

	Av. Dataset1	**Av. Dataset2**	**Av. Dataset3**	**Av. Dataset4**	**Av. Dataset5**	**Av. Dataset6**
Partition Metric	222.12	69.227	105.667	101.2	99.493	107.987
Evaluation Metric	0.958	0.830	0.934	0.896	0.948	0.922
Function Calls	99294.81	42105.42	41916.15	42154.41	42003.8	41916.49

Table 4.10 - Averaging over Dataset.

Looking at the dataset statistics, it appears that Dataset 1 (the mixture of both types of data) produced a higher fitness and independent metric score than the rest and Dataset 3 (the purely VAR data) produced better results than Dataset 2 (the purely DBN data). A reason for this

could be that the VAR data generator produced variables with higher correlations between true dependencies. These may then have outflanked any spurious correlations. It is encouraging to note that the largest dataset, Dataset 1, with a mixture of DBN and VAR data produced such good results. Datasets 4 to 6 which contain a mixture of VAR and DBN exhibit the most variations in the evaluation metric. This is most likely down to the strength of correlations that were reflected in the generated data as well as the existence of spurious correlations.

4.6.4 Sample of Groupings

Table 4.11 shows a selection of groupings learnt from the 3 datasets using the GGA algorithm with differing correlation searches. It can be seen that the majority of variables have been grouped correctly in the all three experiments. In fact, 15 out of the 21 groups have been perfectly recreated. Some of the variables have been placed in a group on their own implying that they are independent when in actual fact there should be some correlation between them and other variables. This could be due to spurious correlations which have prevented the true correlations from being included on the correlation list.

Grouping Method	Original MTS Groupings	Groupings Discovered from Generated Data
EP / GGA **Dataset 1**	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60	0 6 1 2 3 4 5 7 8 9 10 11 12 13 14 15 20 21 22 16 17 18 19 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
Exhaustive / GGA **Dataset 5**	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
Rand Bag / GGA **Dataset 3**	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27	0 1 2 3 5 7 8 9 4 6 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

Table 4.11 - A sample of grouping results from Falkenauer's GGA method along with the original groupings that were used to generate the MTS.

This effect is also evident in the summary tables where the independent metric (which simply measures the distance between the discovered groupings and the original) is higher for some experiments than others but the fitness (which relies on the correlations between variables) is lower. The opposite is also evident in the results. Once again, this is most likely due to spurious correlations between variables in different groups. An interesting result that was found in the DBN data groups was that if a group of variables was incorrectly split into 2 or more groups, then the divide(s) made topological sense when compared to the structure of the DBNs that generated the data. For example, DBN structure 3 in Appendix D contains 10

variables and these consist of variables 13-22 in dataset 1. If the way EP/ Falkenauer algorithm grouped these variables is explored, it can be seen that they placed variables 16-19 in their own group. In some respect it has split the network into relatively independent structures such as the chain of nodes consisting of variables 16-19.

4.7 Results from Real Process Data

In this section the EP / GGA grouping method is applied to a set of oil refinery data. The results of the grouping algorithm are displayed in Tables 4.12. This illustrates the nine groups and the variables associated with each one. The fitness of the individual that represented these groupings was 488.

Feedback from control engineers based on the discovered groupings was very good. Only 5 of the 50 variables were singled out as independent of the others. This was expected as there are a lot of strong relationships amongst most of the dataset's variables. Whilst one of these, variable ID 22, was expected to be included in group H (and a raw data plot supports this hypothesis), the remaining variables upon inspection appear to be extremely noisy. This is the most likely reason for them being excluded from any groups. A small number of variables were found to be grouped in unexpected groups. For example, variable with ID 4, was placed in a small group (Group G) separated from any tray temperatures which was unexpected. However, upon inspection of the raw data plots for this group, the variables were all very highly correlated together.

Group	Variable ID	Variable Name	Chapter 5 Reference
A	2	ABSORB REFLUX TRAY-1	
B	17	ABSORB TAIL-GAS H2 CHROM	
B	27	M/FRACT TOP REFLUX	
C	22	DE-PROP FEED	
D	25	WASH WATER	
E	32	J17-COMP SUCTN. PRESSURE	
F	40	ABSRB STRIPPER BOTTOM	
G	4	ABSORB TAIL-GAS	TGF
G	24	C3/C4 EX CDU3	
G	36	AUTO/MAN STN TO GAS MAIN	
G	37	AUTO/MAN STN TO GIRBOTOL	
H	0	FRESH FEED A-PASS	FF
H	1	FRESH FEED B-PASS	FF
H	3	DEBUT FEED EX ABSORB	BPF
H	5	ABSORB REFLUX TO TRAY-13	SOF
H	6	ABSORB STRIPR WATER LVL	
H	8	REACTOR INLET A	RinT
H	9	REACTOR INLET B	RinT
H	10	SPONGE OIL	SOT
H	12	ABSRB LEAN-OIL TO TRAY11	SOF
H	18	DEBUT O/HDS PCT C2	%C2
H	20	DEBUT OVERHEADS - C2	
H	21	F8 H/CARBON TO ABSORB	
H	23	PROPENE PRODUCT EX J102	
H	26	REFRIDGE A201 TOTAL FEED	
H	28	GAS FLOW TO ABSORB	
H	30	F8 I/STAGE DRUM LEVEL	
H	33	ABSORB SPONGE OIL TRAY11	
H	34	M/F TOP REFLUX PRESS CTRL	
H	35	DEBUT DIF PRESS TRAY1/19	
H	38	J17-COMP SPEED	
H	41	C11/3 INLET	
H	42	J17 SUCTN.	
H	43	J17 I/STAGE	
H	44	J17 DISCH	
I	7	ABSORB PRESSURE CONTROL	
I	11	ABSORB STRIPPER O/HDS	TT
I	13	ABSRB STRIP RBOIL OUTLET	
I	14	E4 OVERHEADS - C3	%C3
I	15	ABSORB TAIL-GAS PCT C3	
I	16	ABSORB T/GAS METH CHROM	
I	19	ABSORB. H2 METHANE RATIO	
I	29	ABSORB BASE LEVEL	
I	31	ABS/STRP TRAY-10	
I	39	ABSORB STRIPPER TRAY-6	T6T
I	45	M/FRACT TOP REFLUX D/OFF	
I	46	M/FRACT TOP TO C06	
I	47	ABSORB STRIPPER FEED	
I	48	ABSORB STRIPPER TRAY-36	
I	49	ABSRB STRIP RBOIL OUTLET	RBT

Table 4.12. The Discovered Groupings from the Oil Refinery MTS. The Final Column Includes Abbreviations for Variables that are Used in Chapter 5.

The most interesting result concerns the two main groups, H and I, which contained a large number of strongly correlated variables, mostly temperatures and flow rates within the main fractionator column. The two groups appear to separate out variables with certain characteristics. For example, group I seems associated more with variables towards the top of the fractionator column such as the higher tray temperatures and the top product quality. Group H, in contrast, is more associated with the lower trays and the bottom product flow and quality. What is more, those variables which are not located in either of these areas of the fractionator but are associated with one or the other, appear in the group where they hold most influence. For example, within this section of data, the variables with ID 0 and ID 1 (the flow rates of the main feed to the FCC) have a strong effect on the bottom product flow (ID 3) and these are included with the associated group (H).

There are of course some exceptions to the lower trays / upper trays split such as variable ID 48, which is a low tray temperature yet is found in group I. However, in general these two groups have fairly consistently separated these two systems. Figure 4.3 and 4.4 highlights the different shaped plots and characteristic features of some variables from two sample groups, *I* and *H*. Looking at the two graphs in Figures 4.3 and 4.4, it can be seen that the variables plotted within each group appear to follow the same general trends.

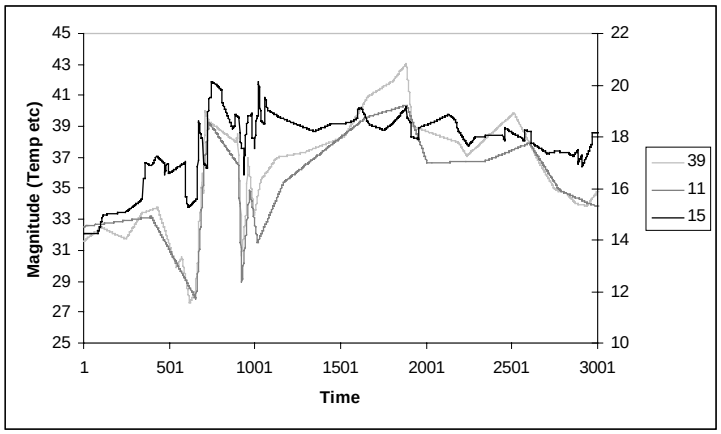


Figure 4.3. Sample variable Plots from Group I

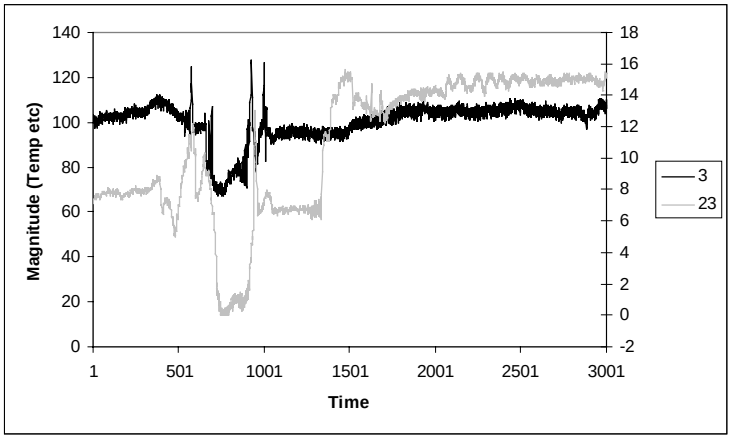


Figure 4.4. Sample variable Plots from Group H

4.8 Conclusions

In this chapter a framework has been outlined with which a high dimensional MTS can be decomposed into smaller dimension MTS which are relatively independent of one another based on the correlation between the variables. This can be very useful in problems where the high dimensionality of a MTS prevents certain algorithms from being applied, for example the generation of Vector AutoRegressive (VAR) models or Dynamic Bayesian Networks (DBNs). The results have shown that whilst the initial search for good correlations to generate the groupings does not have to be exhaustive to produce equally good results, the

best method of grouping search appears to be either a Hill Climb strategy or Falkenauer's Grouping Genetic Algorithm. The results have been very promising on both VAR data and DBN data and, in most cases, the metric used to find the groupings proved robust enough to avoid mistaken groups due to spurious correlations. The chapter has also provided some concrete practical recommendations on the correlation search step of the methodology.

On the real data, the grouping algorithm has produced very encouraging results that will allow a 50 variable oil refinery dataset to be modelled as approximately three independent sub-systems (groups G, H and I). What is more, these groupings have been generated very quickly (a matter of seconds) allowing the algorithm to be used as a pre-processing stage for a model-building algorithm for explaining new data. This rapid model-building algorithm for explaining MTS is the subject of the next chapter.

5 Scaling DBNs for Explaining High Dimensional Time Series with Large Time Lags

In Chapter 3 existing static BN search methods were looked at for the learning of DBN models. However, if these methods are to be used in the learning of DBN models for the rapid generation of explanations of MTS such as the oil refinery data, we must find a way of improving their performance through the use of heuristics and assumptions. This chapter deals with the problem of discovering good DBNs from rapid incoming process data in as little time as possible (see Figure 1.1(a) in Chapter 1) and is based on the work in [Tucker99] and [Tucker2001b].

5.1 Real World MTS

Many real world scenarios involve MTS being generated rapidly, with large numbers of variables and large possible time lags. Data of this nature is evident in many chemical process repositories, as well as other domains such as medical and robot sensor data. Take the oil refinery dataset that we introduced in Chapter 3. This contains three hundred variables and could be expanded to many more if we included variables from other parts of the refinery. It is unlikely that a model incorporating all of these variables could be learnt quickly enough to be of use to control engineers. However, if a method can be found of rapidly learning models of subsets of these variables over large possible time lags, it will be of use to a process engineer who requires an explanation of the current measurements. Chapter 4 has introduced a tool for rapidly decomposing large MTS into several smaller dimensional MTS and these subsets of the original MTS can be used as inputs to such a model building algorithm.

This chapter describes some heuristics which have been taken from previous research papers on learning BNs and introduces some new ones designed specifically for learning DBNs rapidly from MTS process data. We apply these heuristics along with an evolutionary search technique to learning DBNs from synthetic and oil refinery data. We compare our heuristics with other standard searches and use the resulting network structures to generate explanations which are compared to causal diagrams drawn up by control engineers who are familiar with the oil refinery process. The algorithm outlined in this chapter has been adapted from [Tucker99] where it was first introduced.

5.2 Methodology

This section described the general methodology adopted. Section 5.2.1 reiterates the representation that was proposed in Chapter 3. In section 5.2.2 some heuristics are described which are useful in learning a good DBN in as little time as possible. These heuristics are applied to an evolutionary algorithm which makes use of both recombination, the knowledge used by *KGM* in the EP of the last chapter, and other heuristics based on characteristics of the process data. The general methodology is outlined in section 5.2.3.

5.2.1 Representation

We make the same assumption as was made in Chapter 3 during the search method review that a dynamic network contains *no links within the same time slice* (also known as *contemporaneous* links) and represent a DBN as a list of triples. Due to this assumption, each node at time 0, along with its parents, will be independent of the other nodes at time 0. Therefore, we treat the problem of finding a good network structure as a parallel problem of

finding a group of tree structures where each tree consists of a node at time 0 and its parent set in *Q*. The DL of a network structure will simply become the sum of DLs of all the tree structures, DL_i , and the likelihood will be the product of each tree structure's likelihood.

5.2.2 Useful Heuristics

In the context of learning *dynamic* networks, we can apply a standard GA to the representation described in the above section by using the list of triples of each DBN as a chromosome. These triples can then be optimised through the *recombination* and *mutation* operators. However, this technique still takes too long to converge to a good solution as the number of variables increases. Useful heuristics or knowledge are required to speed up the convergence.

i) As there are no contemporaneous links in our proposed representation, each node at time *t*, along with its parents, will be independent of the other nodes at time *t*. Therefore, we can treat the problem of finding a good network structure as a parallel problem of finding a group of simple tree structures where each tree consists of a node at time *t* and all of its parents. So for variable *1* in Figure 2, the tree is represented by the triples *{(0,1,3), (2,1,2), (1,1,1)}.* A change to one tree does not mean the entire structure has to be re-evaluated but only the tree in question.

ii) Observing how the score of an individual triple varies with differing lags shows the resultant curve to be relatively smooth. Figure 5.1 shows an example of the DL of a link with differing time lags using two oil refinery variables - this can be thought of as a Cross

Correlation Function (see Chapter 2) but using DL instead of a correlation coefficient. For this reason a specific mutation has been applied to the lags of a triple (which we call *LagMutation*), and is such that this mutation is based on a uniform distribution with the mean centred on the present lag.

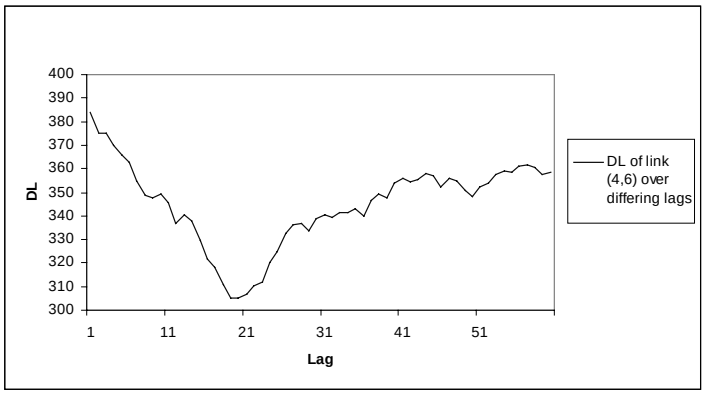


Figure 5.1. The DL of a single link between two oil refinery variables over 60 time lags. Note the relative smoothness of the curve.

iii) Experimentation has shown that autoregressive links with a time lag of one (triples of the form $(a_i, a_i, 1)$) are always the most common in chemical process data. This is most likely due to the relatively smooth nature of the data. For this reason, these links were excluded from possible triples and automatically inserted into the networks before evaluation.

5.2.3 Seeded GA for Search

The sort of algorithm that will be of use to rapid explanation generation is one where a good but not necessarily optimal DBN can be discovered very quickly. The application of a standard GA using the list of triples of each DBN as a chromosome should perform a relatively efficient search over triples through recombination. However, useful heuristics or knowledge may be required to increase efficiency and speed up the convergence. Making use

of pre-processed single link scores through KGM has been shown to be effective [Sahami96, Wong99, Heckerman95, Chow68]. For example, it was used to improve efficiency of an EP in Chapter 3. However, this knowledge is only used each time the operator is applied. The random starting population will be generally poor in quality. If the single link knowledge is to be exploited as soon as possible in order to find better networks in fewer generations, we can seed [Tucker99] the entire first population with links found from the single link analysis. Since the pre-processing of these single links can, itself, take a long time (if the MTS length, dimensionality or *MaxT* is relatively large), it may be preferable to implement an approximate method to find a list of good scoring links. Chapter 4 made use of an algorithm for rapidly finding good correlations in order to group MTS variables. A similar algorithm could be found to find good correlations for seeding the GA. Therefore, the algorithm would be given a head start for the search of good DBN structure in two ways: firstly, by using an approximate method to find a good list of single links rather than scoring the entire set (as the KGM requires); secondly, by exploiting this knowledge in the first population by seeding it entirely with a random selection of good links.

Chapter 4 demonstrated that an EP method is particularly efficient at finding a good selection of links with good correlation, particularly when self-adapting parameters (SAPs) are employed which are able to 'home in' on structures within the dataset (such as over time lags in process data). We can use an approximate algorithm such as an EP for finding good triples to seed an initial population. GAs should be better suited to exploring combinations of these triples through recombination operators. Note that the size of the EP population will

determine the size of the list of good triples. It will be important to consider this value as it will have a large effect on the goodness of triples that are discovered.

If the initial population contains links with good scores as found using an EP, it would be useful if the next stage of search emphasised the recombination of these links. For this reason a uniform crossover operator is used within the GA which will maximise the recombination of the high-ranking single links.

5.3 The Algorithm

See Figure 5.2 for a diagram of the process of seeding a GA with a set of correlations discovered using an EP. This section describes the overall algorithm in full including all the relevant mutation operators.

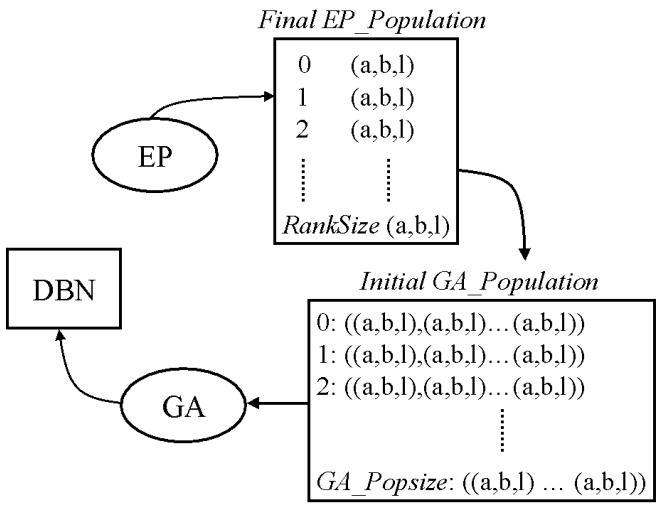


Figure 5.2. The Process of using an Evolutionary Program to Seed the Population of a Genetic Algorithm with good scoring single triples.

I/P *A* (*n*×*N* MTS)
1 If the MTS is not discrete then apply an appropriate discretisation procedure.
2 Initialize *N* to the number of variables and *MaxT* to the maximum possible time lag.
3 Set the initial EP population *EPList* to a random selection of links (a_i, a_j, l)
where $0 \leq a_i < N$, $0 \leq a_j < N$, $1 \leq l \leq MaxT$.
4 For *i* = 1 to *EPGenerations*
5 Score each individual triple using Equation 3.1 or 3.3.
6 Sort *EPList* according to each triple's score
7 Make a copy of each link and apply the *EPMutation* operator to each duplicate
8 Add the mutated duplicates back to the population
9 Remove the lower ranking links until the population is back to its original size, namely *EPListSize*.
10 End For
11 Set a population of triple lists *GAPopulation* of length *random(MaxBranch)* to a selection of links from *EPList*.
12 Construct the network represented by each individual's triple list and set the fitness using Equation 3.1 or the sum of 3.2 and 3.3
13 For i = 1 to *GAGenerations*
14 Sort population according to their fitness
15 Apply *UniformCrossover* depending on *GACrossover Rate* to randomly selected individuals biased on their fitness to generate offspring
16 Apply *LagMutation* to *the* chromosome of the offspring
17 Apply *KGMutate* to *GAMutationRate* percent of the chromosome of the offspring
18 Add all valid offspring to the population and remove the least fittest individuals thus reducing the population to its original size, *GAPopulationSize*.
19 End For
O/P The network structure represented by the individual with the smallest DL/largest marginal probability within the last generation of *GAPopulation*.

Algorithm 5.1. The EP-Seeded GA

Note that the search space for the EP stage will be slightly different to that of the EP search for triples in chapter 5. This is because triples with time lag zero will not be considered valid

and autoregressive triples will now be valid. This results in a search space of $N^2 MaxT$. For the proposed algorithm the following operators were used:

The *UniformCrossover* Operator :

Uniform Crossover [Syswerda89] This operator crosses over subsets of triples within a network structure (see Figure 5.3). As described earlier in this section, each triple in each parent is selected to form part of one of the two offspring based on an unbiased random number generator, having a fifty percent chance of forming part of either offspring's chromosome.

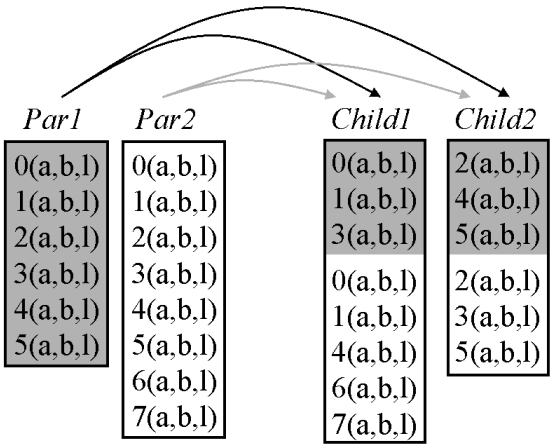


Figure 5.3. Uniform Crossover on the DBN Triple Representation.

The *EPMutation* Operator :

EPMutation is exactly the same as the mutation operator used in Chapter 4 to search for good triples. It uses the notion of self adapting parameters upon elements of a triple to quickly converge to a good selection of links with low DL. When this operator is applied to elements of a triple, $gene_i$, it is mutated according to a normal distribution with the standard deviation

being determined by its self-adapting parameter, σ_i, see Equation 4.1 This parameter is, itself, updated according to Equation 4.2. See section 4.2.1 in Chapter 4 for a full description and set of equations. Notice that rather than using Spearman's rank correlation, a BN metric is used to score triples (either DL or log likelihood).

The *LagMutation* Operator :

This simply mutates the lag of the individual's genes with the probability *Lag Mutation Rate* to a value from a uniform random distribution, *U(lag-T, lag+T).*

5.4 Evaluation

Within this section we investigate two aspects of the learnt DBNs:

i) Efficiency: First of all in section 5.4.1 the efficiency of the standard EP is compared to the proposed algorithm on synthetic datasets and the oil refinery dataset by examining their respective learning curves. The parameters of our algorithm are also investigated as to how they can affect overall efficiency.

ii) Accuracy: Next, in section 5.4.2, the accuracy of the algorithm is tested by looking at structural differences (SD) between networks learnt from the synthetic data and the original network. This is repeated after varying numbers of function calls to see how the quality of the model depends upon learning time. Also in this section, accuracy is investigated using the oil refinery dataset through comparisons of learnt structures with dependency diagrams drawn up

by control engineers and feedback concerning the explanations that have been generated using the discovered networks.

5.4.1 Efficiency

The efficiency of our proposed algorithm was assessed through comparing its learning curves to the standard EP on synthetic data and the oil refinery MTS. The refinery data consisted of 1000 datapoints over 11 variables that were discretised into four states. For these experiments, the algorithms were parameterised as shown in Table 5.1 below. Notice that the number of calls during the pre-processing stage, *c*, was varied between 20% of the total search space and 100%. In the 100% case, the time taken for pre-processing is identical to that of the pre-processing stage of the standard EP algorithm. It must be noted that *ListSize* and *c* can be set according to the available time and required accuracy of the final DBN. For all these experiments *MaxT*=60 and *MaxBranch*=3.

	c	*PopSize*	*GA PopSize*	*ListSize*	*GA Crossover Rate*	*GA Mutation Rate*	*Lag Mutation Rate*
Standard EP	100%	10	-	100%	-	-	-
EP-Seeded GA	20 / 100%	-	10	2.5%	0.8	0.1	0.1

Table 5.1. The Parameters for EP-Seeded GA and Standard EP

Figures 5.4(a) to 5.4(f) show the learning curves of the methods: standard EP and the proposed EP-Seeded-GA with *c*=20% and *c*= 100%. Notice that EP-Seeded GA where *c* is 100%, which takes the same amount of time to pre-process as the standard EP, not only begins with better scoring network structures but continues to improve at a similar gradient to the standard EP method. When the curves finally do meet, they generally converge at a

similar rate. In contrast, the EP-Seeded-GA with *c*=20% starts off with a score better than standard EP. However, as the function calls increase this method slows down in convergence rate. In fact on the real dataset at the later stages of the experiments, it is overtaken by the standard EP.

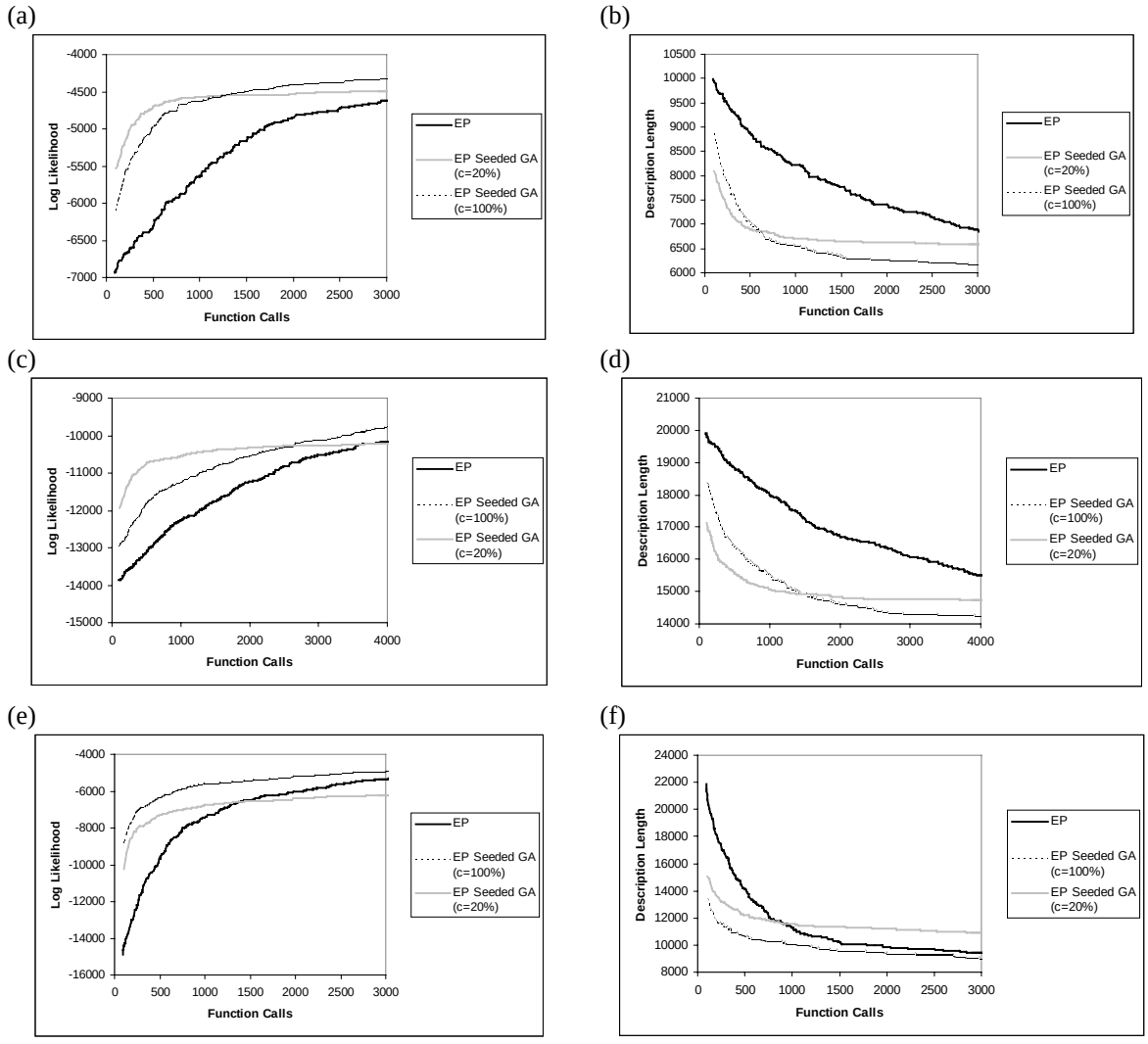


Figure 5.4. Performance on Synthetic Datasets: (a) (b) *N=10, MaxT=60;* (c) (d) *N=20, MaxT=60*; and Oil Refinery Datasets: (e) (f) *N=11, MaxT=60.*

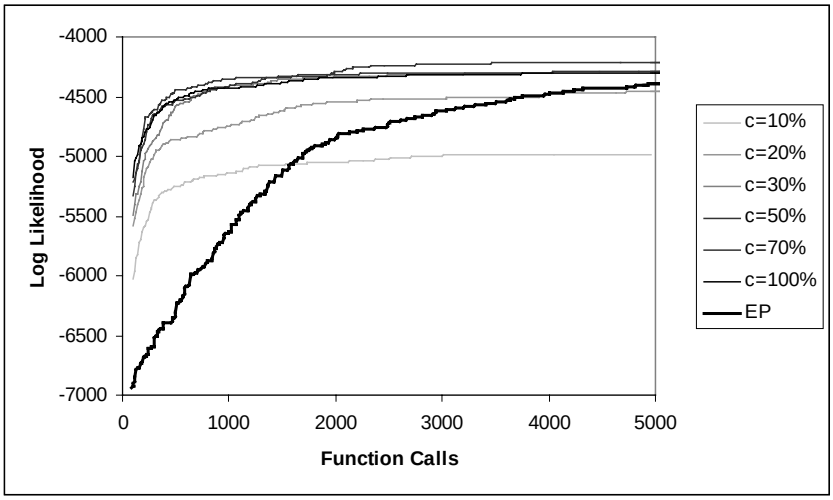


Figure 5.5. Performance on Synthetic Dataset with *N*=10 and *MaxT*=60 with varying number of calls, *c*, in the EP-Seeding Stage of EP-Seeded-GA.

Figure 5.5 shows how the number of calls in the EP seeding stage affects the overall efficiency of the EP-Seeded-GA when learning DBN structure. The number of calls, *c*, is varied between 10% and 100% of the entire search space. An interesting feature of this graph is that the performance when *c* is 30% is almost identical to that when *c* is 100%.

To sum this section up, on larger MTS the pre-processing stage (learning the single link information) requires much more time and so if good networks are required *rapidly*, it appears that the approximate approach utilised by EP-Seeded-GA with a lower value of *c* (around 30%) is the most suitable. If time is not as expensive, the EP-Seeded GA with *c*=100% is recommended as it takes the same amount of time to pre-process the single links as the standard EP and is more likely to converge quicker.

5.4.2 Accuracy

In this section the resulting structures generated from the synthetic datasets after varying number of function calls (FC) were investigated by calculating the Structural Difference (SD). The SD is the summation of all those links that were found within the resulting structures but should not have been due to spurious correlations and implicit dependencies (see Chapter 2 for a description of these), and all links that were missing from the resultant structures but which should have appeared. The smaller the SD the better the structure is deemed to be. If the time lag of a parent is out by three or fewer, then it is not deemed to be different from the original network. This value was arrived at as an explanation that was incorrect in the time aspect by three minutes or fewer would not normally affect the overall quality. What is more, discretisation of the data may affect the accuracy so that precise time lags may be shifted a few minutes in either direction. Here we only show results from this analysis using the log likelihood metric. DL scores produce almost identical results.

FC	EP-Seeded-GA		Standard EP	
	N=10 *MaxT*=60	*N*=20 *MaxT*=60	*N*=10 *MaxT*=60	*N*=20 *MaxT*=60
100	16	25.6	12.5	11.2
500	14	22.6	19.2	15
1000	12.8	20.4	16.4	21
2000	11.2	22	18.4	26.6
5000	9.2	20.6	24.6	31.4
10000	7.6	19.2	8.7	31

Table 5.2. The Average Structural Differences (SD) between the original DBN and the discovered DBN using EP-Seeded-GA and Standard EP with Log Likelihood after varying numbers of FC.

The SD analysis (Table 5.2) shows a surprising result in the standard EP algorithm. Whilst the SD of the EP-Seeded-GA generally decreases as function calls increase as expected, this

is not evident in the standard EP. In fact, for *N=20*, the SD generally increases. This is most probably due to the EP initially finding structures that produce relatively good DBN metric scores against the dataset even though they bear little relation to the original generating structure. This may be through the discovery of the correct links at the expense of also finding spurious correlations or implicit dependencies. Recall from Chapter 2 that a spurious correlation is a dependency that appears to exist between two nodes due to a common parent between the two nodes while an implicit dependency is one which appears to exist due to indirect causes

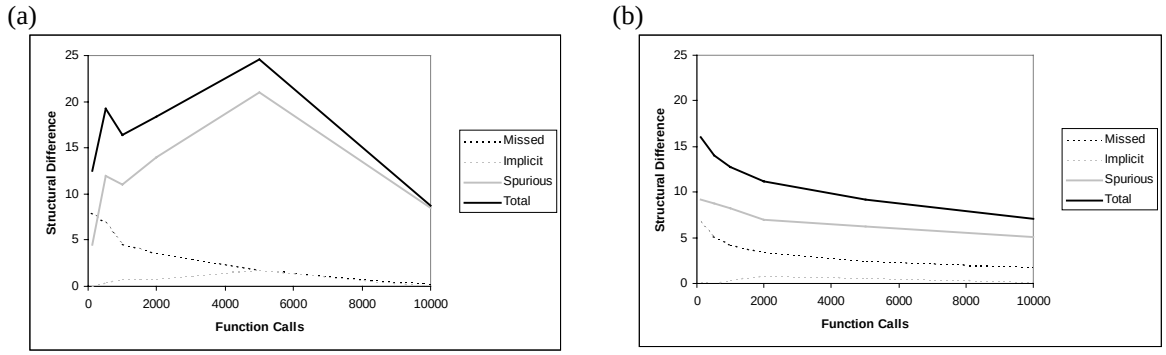


Figure 5.6. Breakdown of Average Structural Difference on Synthetic Dataset where *N=10* and *MaxT=60* using (a) Standard EP and (b) EP-Seeded-GA.

The hypothesis that spurious correlations account for the rise in SD for standard EP is supported by Figure 5.6(a). This shows the breakdown of the SD into implicit dependencies, spurious correlations and missed links. The actual number of missed links decreases as function calls progress. However, the number of spurious and implicit correlations actually increases up to 5000 function calls before falling. It appears that the EP-Seeded-GA avoids this growth with all elements of SD generally decreasing (see Figure 5.6(b)).

For the oil refinery dataset, an analysis is required of how accurately the learnt network structure represents the sort of relationships an expert would find. This is done by asking control engineers, who have extensive knowledge of the refinery process and data, to produce some dependency diagrams that represent the expected relationships between some of the variables in the oil refinery MTS (see Figure 5.7). These diagrams are then compared to explanations generated using the discovered networks. To generate these explanations, evidence about a subset of variables at various time slices is entered into the network and inference is performed on the network.

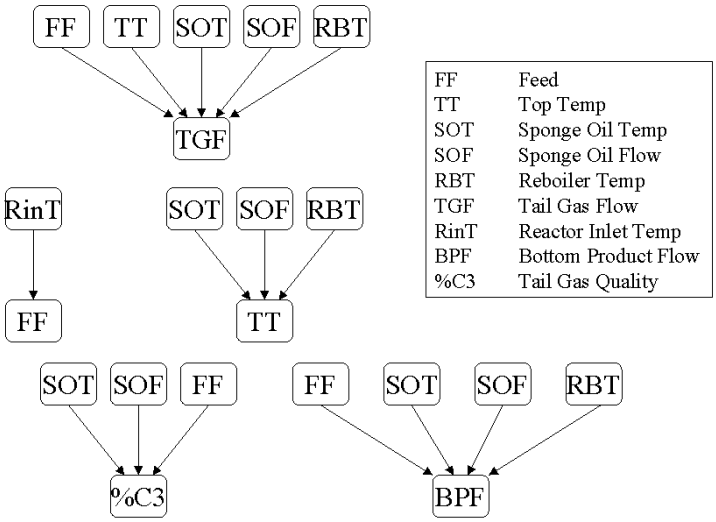


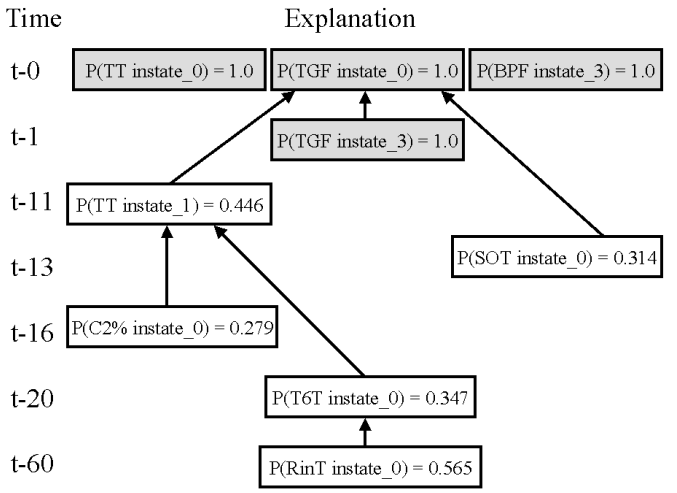
Figure 5.7 - Sample Dependency Diagrams constructed from advice of Control Engineer

Figure 5.8 shows some of the explanations that were generated from the learnt structures using two oil refinery datasets, one with $N = 11$ and the other with $N = 20$. The algorithm used to learn these structures was EP-Seeded-GA with *ListSize* being 2.5% of the entire search space, *c* being 20% of the search space, and the algorithm was stopped after 1500 and

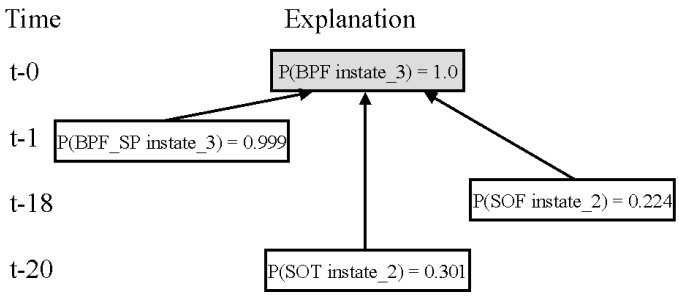
2000 function calls for the 10 variables and 20 variables dataset, respectively. Shaded boxes in Figure 5.8 represent observed nodes.

It is very encouraging to note that the current algorithm detects all of the relationships within Figure 5.7 correctly except for those limited by *MaxBranch* being set to 3. For example, the explanation in Figure 5.8(a) has captured TGF as being dependent upon SOT and TT. However, the algorithm generates more explanations than those found in the dependency diagram. This is to be expected since the diagram is not meant to be exhaustive in that it only captures some of the obvious relationships that should exist in the dataset. From Figure 5.8(b), we can see that BPF is affected by three variables but mostly by its controller setpoint BPF_SP. This is discovered from the data (the probability of BPF_SP being in state 3 if BPF is in state 3 in the next time step is 0.999). However, if we observe BPF_SP as being in state 0 at *t-1* (Figure 5.8(c)), we see an increase in the probability of other variables being the cause of BPF's current state.

(a)



(b)



(c)

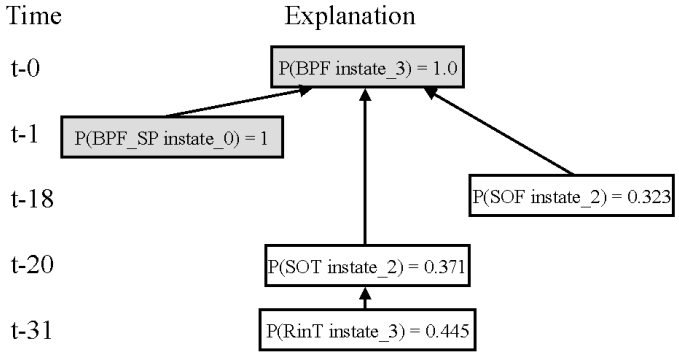


Figure 5.8 - Sample explanations generated using the refinery data. Shaded blocks represent observed variables.

However, it must also be pointed out that there were a few relationships found within the network structure that were known to be false, leading to some incorrect explanations. These are likely to have occurred for various reasons. For example, a false link in the opposite direction of causality, is most likely caused by spurious correlations. These are likely to occur due to the smoothness of the Cross Correlation Function (CCF) between two variables (see Figure 5.1) where a strong dependency from variable *4* to variable *6* could produce a strong correlation in the CCF in the opposite direction, from *6* to *4*. Other reasons include loss of information during discretisation or if some related variables are missing. For example, in Figure 5.7 FF and TT affects TGF, but if TGF is missed out of the dataset, it is quite possible that a dependency will be found between FF and TT. It is, therefore, important to ensure that all of the relevant variables are included within the dataset and a good discretisation policy is adopted.

5.6 Conclusions

The learning of dynamic probabilistic models with large time lags is an important issue, not only for complex process applications but also for many AI problems (e.g. learning domain behaviour for robot navigation, data-mining for temporal sequences, learning to control a complex plant). Chapter 3 investigated different existing search algorithms for static BNs which had been adapted to learn DBNs. It found that most methods suffered from various problems such as local maxima and slow convergence. Within this chapter the use of evolutionary methods have been investigated in order to improve upon the existing search strategies. The number of possible network structures can be huge, even when dealing with a small number of variables due to the consideration of large possible time lags. Tested on

synthetic datasets and oil refinery data, the proposed algorithm has demonstrated success in managing this complexity and in producing timely, good quality explanations.

In the next chapter, the focus will turn from trying to generate explanations rapidly, to the subject of learning different models from large repositories of historical data off-line, where the dependencies between variables can change over time.

6 Detecting Dependency Changes within a Multivariate Time Series whilst Learning DBNs

When learning from process datasets with many more time points, new problems arise. For example, in many systems such as an oil refinery, dependencies between variables may change over time. The dependencies can be affected by the way operators control the processes and by the different products being refined at the current time. If these changing dependencies are not taken into account any model that is learnt from the data will average over the different dependency structures. This feature will not only apply to chemical process systems but also to many other dynamic systems in general.

There has been extensive work in the modelling of time series with changing dependencies (or 'switching states'). For example Hidden Markov Models have been used to model the hidden states of a system which can change over time [Gharamani99, Shumway91, Kim94, Settimi99]. However, these models can contain huge numbers of parameters causing problems for very high-dimensional data with large time lags, and do not necessarily aid in the understanding of the underlying processes. The aim of this chapter is to tackle the problem of learning models from the vast data repositories that many chemical processes have generated over the years (see Figure 1.1(b)) by incorporating changing dependencies but at the same time remaining transparent so that the resulting models and explanations can account for these changes.

In Section 6.1, the Dynamic Cross Correlation Function is introduced which is used to analyse the changes in dependency structure between two variables within MTS. Section 6.2 outlines the methods and algorithms investigated to incorporate the changing dependency structures into the DBN paradigm described in the previous chapters and 6.3 details the experiments and their results when applied to synthetic and real MTS data. Finally in 6.4, some explanations are generated from the resulting structures and conclusions are drawn in 6.5.

6.1 The Dynamic Cross Correlation Function

During the analysis of MTS we have found it useful to explore how the Cross Correlation Function (CCF) between two variables changes over time. Recall from Chapter 2 that the CCF is used to measure the correlation between two time series variables over varying time lags by time shifting the data before calculating the correlation coefficient. Analysis of data with changing dependencies has involved developing a Dynamic CCF (DCCF), $\rho_{a_i a_j}(l, t_s, t_f)$, whereby the CCF is calculated over lags, l, between two variables, a_i and a_j, for a window of data delimited by t_s and t_f, and moved over the MTS by incrementing the window position by set amounts (see Figure 6.1). The calculation of the $\rho_{a_i a_j}(l, t_s, t_f)$ is formalised in Equations 6.1 and 6.2.

$$\rho_{a_i a_j}(l, t_s, t_f) = \frac{\gamma_{a_i a_j}(l, t_s, t_f)}{\sqrt{\gamma_{a_i a_i}(0, t_s, t_f)\gamma_{a_j a_j}(0, t_s, t_f)}} \quad 6.1$$

$$\gamma_{a_i a_j}(l, t_s, t_f) = Cov(a_i(t_s, t_f), a_j(t_s + l, t_f + l)) \quad 6.2$$

where $a_i(t_s,t_f) = \{ a_i(t_s), a_i(t_s +1), \ldots, a_i(t_f)\}$ and t_s and t_f are determined by the window position and length, and *Cov(x,y)* returns the covariance function of *x* and *y*. If $\rho_{a_i a_j}(l,t_s,t_f)$ is calculated for all lags and window increments (the process is illustrated in Figure 6.1), a surface plot of the correlations is generated, known from now on as the DCCF, such as the example in Figure 6.2.

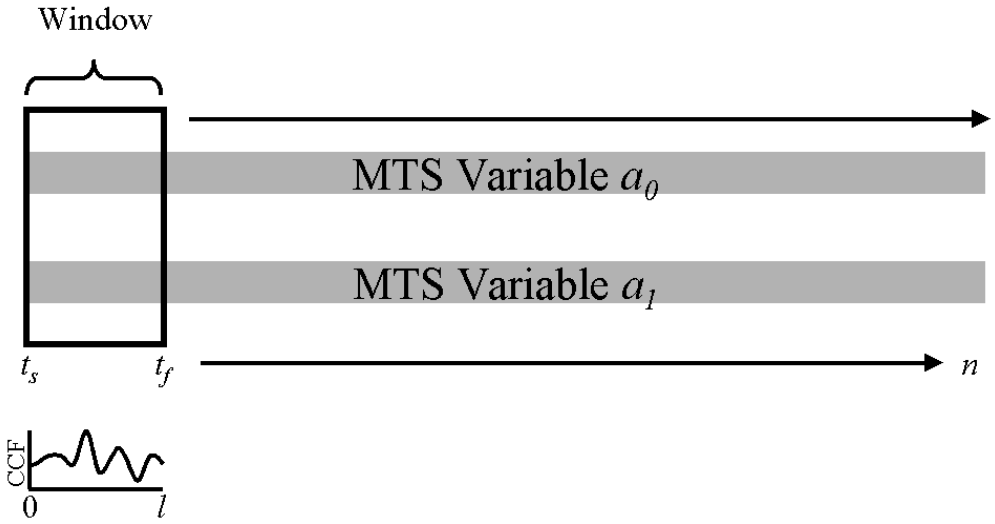


Figure 6.1. Generating the Dynamic Cross Correlation Function (DCCF).

The algorithm for generating such a plot is a simple iteration over window positions and time lags as described in Algorithm 6.1.

```
I/P    a_i , a_j , win_len ,win_jump, MaxLag
1      pos = 1
2      While pos < n-win_len
3          t_s = win_jump x pos,  t_f = t_s + win_len
4          For l = 0 to MaxLag
5              Set DCCF(l, pos) = ρ_{a_i a_j}(l,t_s,t_f)
6          End For
7          pos = pos + 1
8      End While
O/P    DCCF
```

Algorithm 6.1 - Constructing the DCCF Surface Plot.

Whilst the DCCF will not be able to tell us about the more complex relationships within the MTS that involve more than two variables, it will assist us in making preliminary judgements about where likely dependency changes occur. Each horizontal cross section of a DCCF is a distinct CCF. Notice how the surface plot in Figure 6.2 changes as the window position progresses. For example there appears to be a weak correlation between the variables from window position 10 to 20 with time lags of between 5 and 35 (indicated by *'pos1'*) , and another stronger area of correlation between window position 60 and 70 with a time lag of between 5 and 25 (indicated by *'pos2'*).

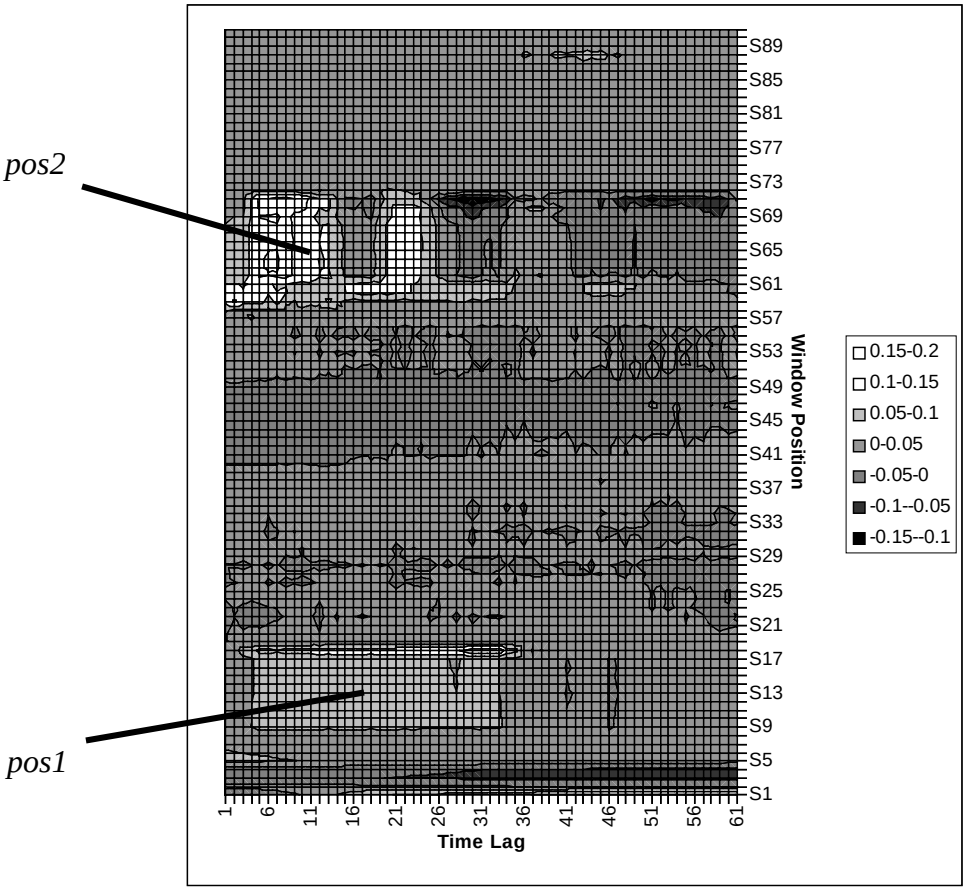


Figure 6.2. An example DCCF applied to two variables from the oil refinery dataset (TT / TGF).

6.2 Learning DBNs from MTS with Changing Dependencies

Controllers are used in the oil refinery plant to control how a variable behaves in respect to other measurements (see section 2.1.3). For many variables, these controllers are not measured or do not even exist (approximately 44% of the FCC data variables are controlled). However, in some sense, all of the variables will be controlled in differing ways depending on how the process is being operated. We can therefore model these behaviours as 'hidden' controllers. For example, sometimes a variable is directly controlled by an engineer (i.e. when a controller is in 'Manual') and a hidden node can be used to model the way that the engineer controls that variable.

The problem of modelling changing dependencies in MTS is that on one hand we know part of the structure (the dependency between each variable at time *t* and the hidden controller for that variable) but we need to calculate the parameters; on the other hand we do not know the remaining part of the structure (the parents of each node at time *t*) but the parameters for these can be trivially calculated (we assume no missing data on all variables except controllers). Unfortunately, both these problems are fundamentally intertwined, with the parents of each node at *t* affecting the parameters of the hidden controller and the parameters of the hidden controller affecting the parents of node at *t*. What is required is some method for maximising the likelihood of the DBN through the manipulation of both these features.

A hill climb search is investigated within this chapter when applied to a representation that has been extended from that used in the previous chapters to learn DBN models. We have also used this representation with the Structural Expectation Maximisation (SEM) procedure

[Friedman97/98b] described in Chapter 2. However, due to the disappointing results from this method we have chosen to focus on the simple hill climb in this chapter. The results of the SEM experiments can be found in Appendix G.

In the case of modelling process data with changing dependencies, the hill climb search can be simplified due to certain features of the process data and this is discussed in the next section.

6.2.1 Representation

By including a *hidden controller node* as a parent for each node at time *t* representing variable *i*, which we will refer to from now on as $Opstate_i$, the dependencies associated with that variable can be controlled (see Figure 6.3). This hidden variable does not have to be sought after and only the values for this node need to be discovered. These *OpState* nodes will determine how the variables in the plant behave based upon their current state.

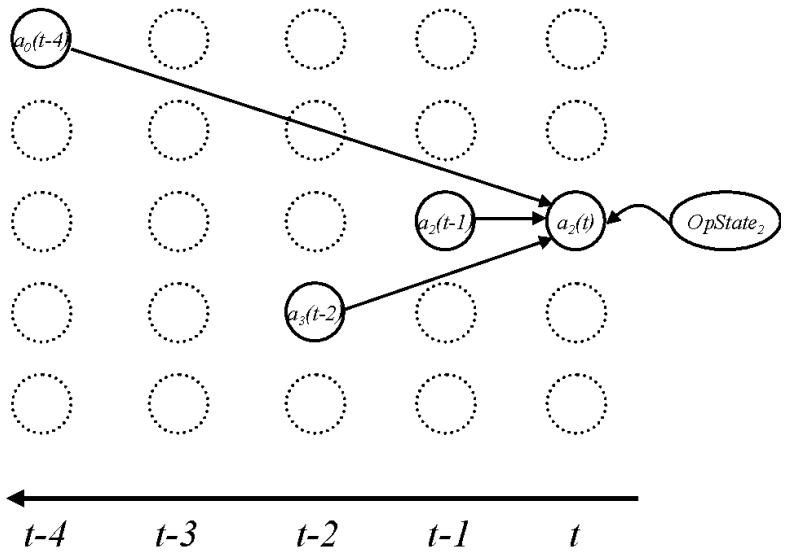


Figure 6.3. Using a Hidden Variable, $OpState_2$, to act as a 'Controller' for variable a_2 at

time *t*. Each variable, *i* is assigned an $OpState_i$.

The representation for structure is precisely the same as previous chapters using a list of triples. However, for learning models with changing dependencies, the *OpState* node is automatically inserted before evaluation. We will call the list of triples *DBN_List* for the remainder of this chapter.

The switching of *OpState* is not likely to happen very often in chemical process data and so the relative stability of these variables can be used to speed the convergence of a hill climb. For this reason, the hill climb will use a list of pairs *(state,position)* to represent the switches in *OpState* where *state* represents its new state and *position* represents the position of change in the MTS. The pair with the lowest value of *position* determines the starting state of *OpState* from position 1 until the next change. For example, taking the pairs:

{(1,230), (0,500), (2,750), (1,900), (0,1250)}

OpState will be set to state 1 from MTS positions 1 to 499 (determined from the first pair because 230 is the lowest *position* value), then state 0 from 500 to 749, state 2 from 750 to 899, state 1 from 900 to 1249 and state 0 from 1250 until the end of the MTS. This list will be called *Segment_List*. A heuristic-based hill climb procedure is employed by making small changes in *DBN_List* and the *Segment_List* for each *OpState* variable. The algorithm is described in the next section.

6.2.2 The Hidden Controller Hill Climb Algorithm

In order to search over the DBN structure and the parameters of *OpState* for each variable, the Hidden Controller Hill Climb (HCHC) algorithm uses the representation and assumption

described in section 6.3.1. This is illustrated in Figure 6.4(a) where *DBN_List* is used to generate the structure of the DBN and the *Segment_Lists* are used to determine the parameters of each variable's *OpState*. The resulting network can then be scored using log likelihood as defined in Chapter 3.

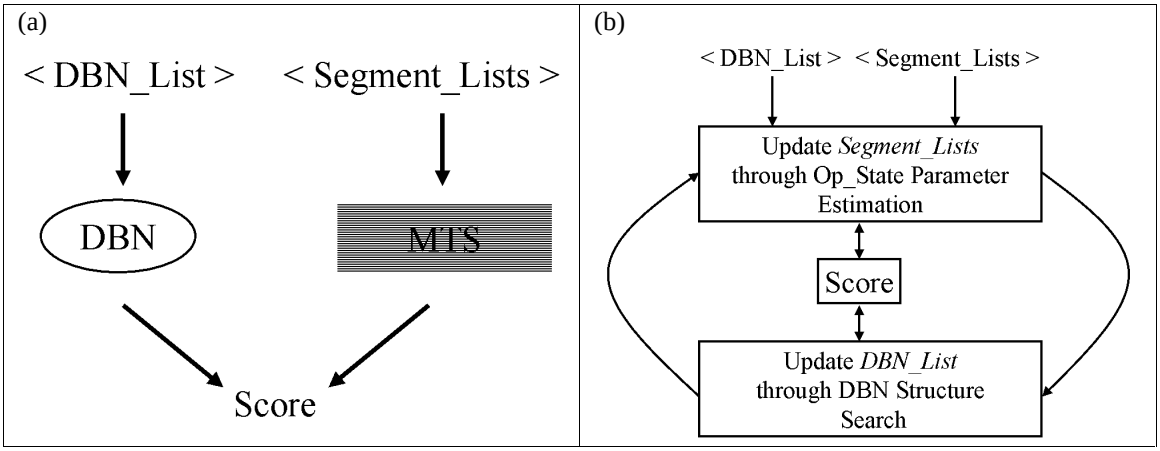


Figure 6.4. (a) The Procedure for scoring the Current Model. (b) The HCHC Algorithm for Segmenting Process Data and learning DBN structure.

Figure 6.4(b) shows the HCHC process. HCHC takes as input the initial random *DBN_List* and a *Segment_List* for each *OpState* node, along with the MTS, *A*, the maximum time lag, *MaxT*, and the number of iterations for the structure search, *DBN_Iterations*. It then applies a standard hill climb search to the *Op_State* parameters by making small changes to each *Segment_List* and keeping any that result in an improved log likelihood score. Structure search is then carried out by repeatedly making small changes to *DBN_List* and keeping any changes that improve log likeliood. After structure search is repeated *DBN_Iterations* times, the entire process is repeated, returning to the *Op_State* parameter search. The search ends when the function calls, *FC*, reach some pre-defined value, *Iterations*. The algorithm is defined formally in algorithm 6.2.

```
I/P   A, MaxT, DBN_Iterations, Random DBN_List, and N
      Segment_Lists
1     FC = 0
2     Using the current DBN_List and Segment_Lists construct
      the DBN and parameterise
3     Best_Score = log likelihood of DBN (Equation 3.1)
4     Repeat
          OpState Parameter Search
5         For i=1 to N
6             Apply random change to the Nth Segment_List
7             Using the current DBN_List and Segment_Lists,
              construct the DBN and parameterise
8             If log likelihood of DBN > Best_Score Then
9                 Best_Score = log likelihood of DBN Else
10                Undo change to Nth Segment_List
11            End If
12        End For
          Structure Search
13        For j=1 to DBN_Iterations
14            Apply random change to DBN_List
15            Using the current DBN_List and Segment_List
              construct the DBN and parameterise
16            If log likelihood of DBN > Best_Score Then
17                Best_Score = log likelihood of DBN Else
18                Undo change to DBN_List
19            End If
20        End For

21        FC = FC + 1
22    Until Convergence or FC>Iterations

O/P   The Final DBN_List and N Segment_Lists can be used to
      construct the resultant DBN and Op_State parameters
```

Algorithm 6.2 - The HCHC Segmentation Algorithm.

DBN_List and the *N Segment_Lists* are output and used to generate the final DBN structure and *Op_State* parameters. For example, given the *DBN_List* and five *Segment_Lists* in figure 6.6 we can construct the DBN structure with a link from a_1 to a_0 with a time lag of 8, from a_0 to a_1 with a time lag of 5, and so on. We can then learn the parameters for the observed

variables using the MTS data. The *Op_State* variables, however, must be reconstructed using the *Segment_Lists*, so referring to Figure 6.6, the state of $OpState_0$ will be set to 0 for all time points from 0 to 1998 and for the remainder of the series it will be set to 4. Once all *Op_State* variables have been constructed, the DBN parameters can be learnt.

6.3 Experiments

The HCHC algorithm has been tested on both synthetic and real-world datasets. The remainder of this chapter documents the results of these experiments and the analyses of results.

6.3.1 Results from Synthetic Data

Synthetic data has been generated from hand-coded DBNs using stochastic simulation (see section 2.1.2). The MTS datasets consisted of between five and ten variables with time lags of up to 60 time slices. In order to incorporate changing dependencies within the datasets, different DBN structures were used to generate different sections of the MTS. Essentially, several MTS were appended together, having been generated from DBNs with varying structures and parameters. For example, MTS1 consists of three MTS with 5 variables appended together, each of length 1000. Characteristics of the datasets are described in Table 6.1 and the make-up of MTS3 is illustrated in more detail in Figure 6.5.

	MTS 1	MTS 2	MTS 3
MTS Dimensionality	5	10	5
Total MTS Length	3000	3000	2500
Segment Length	1000	1000	500
Maximum Time Lag (*MaxT*)	5	60	25
Number of Segments	3	3	5

Table 6.1. Details of the Synthetic Data with Changing Dependencies.

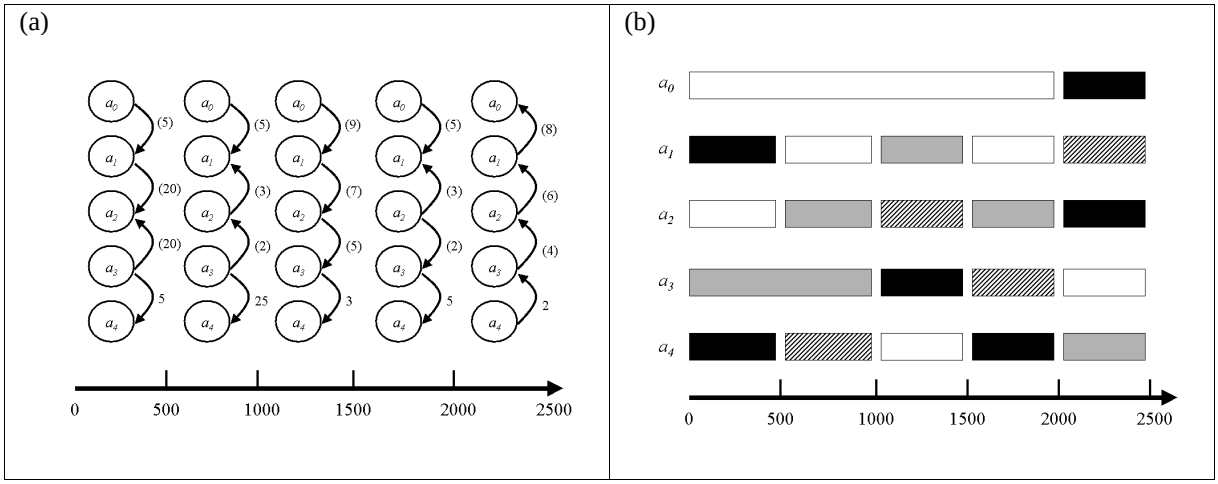


Figure 6.5. (a) Each DBN is displayed corresponding to the different segments of MTS 3. The numbers in parenthesis denote the time lag for that particular link. The positions of state change are included on the lower axis. (b) illustrates the positions of dependency change generated from the DBNs in (a).

The experiments involved applying the HCHC search procedure to learn the dependency structure and segment the data for each variable according to its *OpState*. The resultant structures and segmentations at convergence were then compared to the original structures that generated each segment of the MTS by calculating the Structural Differences (SD), as used in Chapter 5. Segmentation error was calculated by dividing the distance from the correct segmentation point by the size of the segment. This was repeated for each segment in each variable for all experiments and an average taken. The number of missed and spurious segmentations was also recorded. A segmentation was deemed spurious if the error was

greater than 10% of the segment length. This value was agreed with control engineers as being of reasonable size to adversely affect explanation.

The results from using synthetic MTS have been very encouraging. Nearly all the dependencies of the original networks were recovered. Segmentation varied from experiment to experiment, but the distance between the discovered dependency change and the actual change was usually around 15 time points on average. Both dependencies and segmentations were normally found within 250 iterations. Table 6.2 contains the SD between the original and the discovered structures. It shows the average number of missed dependencies, spurious correlations, implicit dependencies and segmentation errors for each dataset. Also included are the number of links in the original DBNs and the length of each segment of data.

	MTS1	MTS2	MTS3
Number of Original Links	12	26	16
Spurious	2.3	2.9	4.0
Implicit	2.3	1.0	0.4
Missed	1.0	2.8	1.4
Total SD	5.6	6.7	5.8
Original Segmentation Length	1000	1000	500
Segmentation Error	15.89	16.08	14.157
Missed Segmentations	0.6	0.0	1.2
Spurious Segmentation	0.9	0.5	0.8

Table 6.2. Structural Difference Results using HCHC

The table illustrates that the quality of segmentation is generally very high. For all datasets the number of spurious, missing and implicit links are small in comparison to the number of actual links correctly discovered. Segmentation is also very good. The average error is a tiny proportion of the length of a segment. Very few segments were missed (0.6 , 0.0 and 1.2 on

average for MTS1, MTS2 and MTS3 respectively), and the most likely reason for those that were is that the change had very little impact on the data. For example, a change from variable a_0 to a_1 with lag of 3 to another link from a_0 to a_1 with a lag of 7 may cause the segmentation to be missed. A sample *DBN_List* and the *Segment_Lists* discovered for each variable from MTS 3 is shown below and, referring to Figure 6.5, indicates a very good match to the original structures and segmentations:

DBN_List: {(1,0,8), (0,1,5), (0,1,9), (2,1,3), (2,1,6), (3,2,2), (1,2,7), (3,2,20), (4,3,3), (2,3,5), (3,4,3), (3,4,25), (3,4,5)};

Segment_List for $OpState_0$: {(0,0),(1998,4)}

Segment_List for $OpState_1$: {(0,0),(500,1),(1022,2),(1487,3),(1998,4)}

Segment_List for $OpState_2$: {(0,0),(999,2),(1491,3)}

Segment_List for $OpState_3$: {(0,0),(996,2),(1475,3),(1995,4)}

Segment_List for $OpState_4$: {(0,0),(498,1),(981,2),(1502,3),(1997,4)}

Figure 6.6. Resulting DBN and Segments on MTS 3 using HCHC.

Using these lists we can construct our DBNs by adding the links to each node depending upon its set of parents and segmentations. For example, for variable a_0, only one parent has been found, (1,0,8), and the partitions discovered for this variable signify a change at around the 2000th position in the MTS (1998 to be precise) where the dependency begins to take effect. Referring to the original structures we only find one example of this dependency - in the 5^{th} segment which does indeed occur between time points 2000 and 2500 in MTS 3. Overall, the results for each variable reflect the changing dependencies found in the original networks very closely. Some time lags and segmentation positions vary from the original structures but not significantly, and 13 of the 16 original dependencies have been recovered.

We now make use of the DCCF in order to further analyse the results of the HCHC on MTS 3. Whilst this is not really necessary for synthetic data where the original network structures are available, it will be useful when analysing real world data where these structures are unknown. Focussing on variable a_4 from MTS 3, it can be seen how the dependencies with variable a_3 in MTS 3 are evident in its DCCF. Figure 6.7(a) shows a DCCF for these two variables. Recall that this surface plot can be viewed as a set of standard CCFs which are appended together where each horizontal position in the surface represents a CCF for that window position.

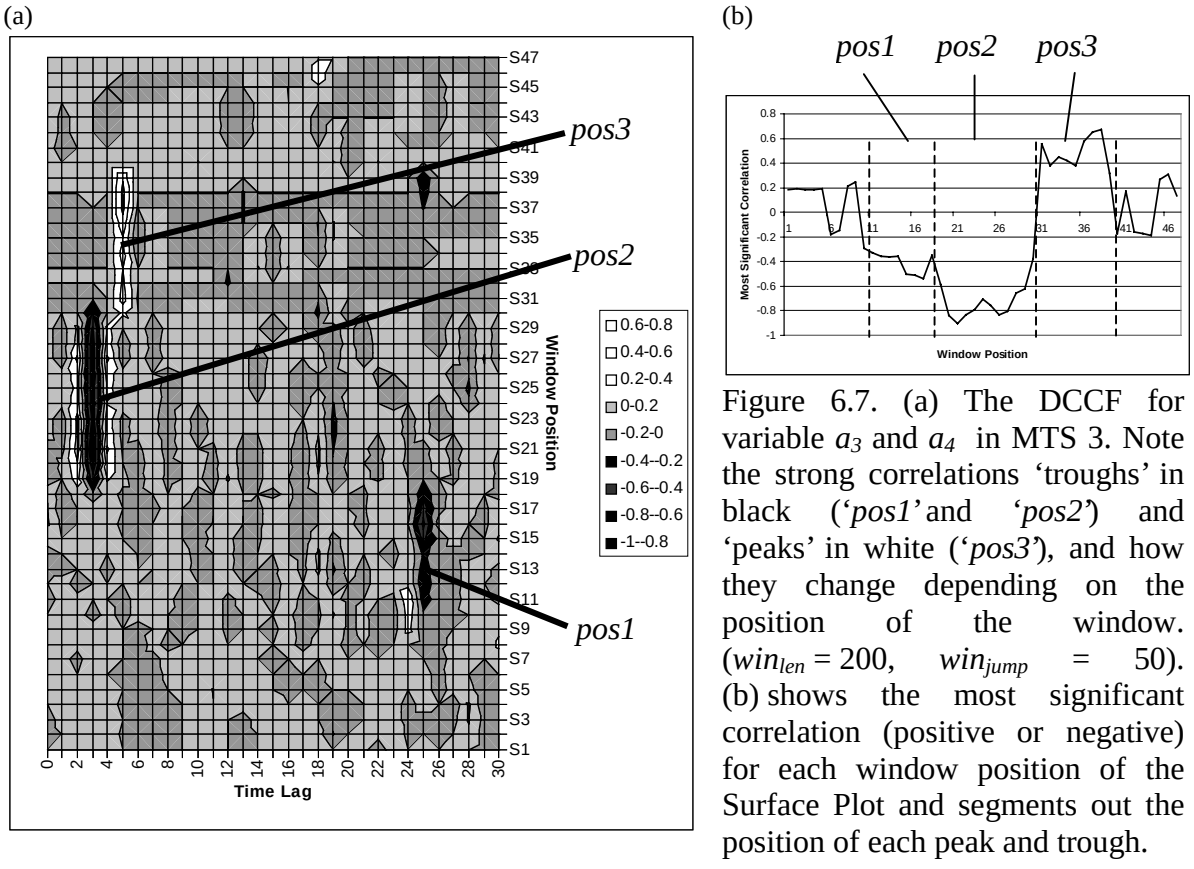


Figure 6.7. (a) The DCCF for variable a_3 and a_4 in MTS 3. Note the strong correlations 'troughs' in black ('*pos1*' and '*pos2*') and 'peaks' in white ('*pos3*'), and how they change depending on the position of the window. (win_{len} = 200, win_{jump} = 50). (b) shows the most significant correlation (positive or negative) for each window position of the Surface Plot and segments out the position of each peak and trough.

In this example, the size of the window, win_{len}, was 200 time points and with each successive CCF calculation the window was shifted along the MTS 50 time points, win_{jump}, the

maximum time lag, *MaxLag*, calculated for each CCF was 30. Figure 6.7(b) shows a graph of the most significant correlation for each window position in the DCCF. In other words the maximum absolute correlation for each CFF as the window travels along the MTS. In Figure 6.7(a) note the varying peaks (in white) and troughs (in black) which signify stronger positive and negative correlations, respectively. The DCCF between variable a_3 and variable a_4 in Figure 6.7 shows five dependency states. Firstly there is no apparent correlation from a_3 to a_4 for window positions 1-10, then an inverse correlation occurs (the black trough denoted by 'pos1') with a lag of 25 in window positions 10-20, this lag then switches to 3 for positions 20-30 (denoted by 'pos2'), then a positive correlation can be seen (white area denoted by 'pos3') with a lag of 5 in positions 30-40, and in the final sections of data (positions 40-50), no correlation from a_3 to a_4 can be found. Note that these correlations correspond well to the *DBN_List* and the *Segment_List* for a_4, discovered for MTS 3 - there are five different segments appended together which relate to the section before *pos1*, the section during *pos1*, the section during *pos2*, the section during *pos3*, and the section after *pos3*. In addition, the time lags between each link from a_3 to a_4 tie in with the time lags relating to the significant correlations in *pos1*, *pos2* and *pos3*.

6.3.2 Results from Process Data

The application of the HCHC segmentation algorithm to the process data involved making use of some of the available controller information. Recall from Chapter 3 that a controller can be in three states: Manual, Automatic and Cascade. The different modes of controller state should have an effect on what relationships exist between the variable in question and others in the dataset. The following set of experiments involved applying Algorithm 6.2 to a

selection of oil refinery variables and comparing the discovered segmentations with the DCCFs between certain variables.

It has been found upon examination of the raw data that many changes in the observed controllers states have little effect upon the relationships within the data. This is likely to be because when a controller changes from, say, Auto to Manual, the same control is kept over the variable (although the way this is done may differ). This means that many of the discovered segmentations do not concur with the changes in the observed controller states. There are, however, some controller state changes which do result in dependency changes and concur with the discovered segmentations. Although the vast majority of segmentations are not found in the measured controller data, when we look at the raw data, obvious changes in dependency structure have occurred due to genuine 'hidden' causes. These are documented in the remainder of this section.

In Figure 6.8, the most significant correlation graphs for each window position of the DCCF are shown for MTS variable TGF with each of its three discovered parents (the full DCCF from which these are constructed can be found in Appendix F). Super-imposed on these graphs are the discovered segments. It can be seen that there is generally a fluctuation between no and strong negative correlations between TGF and A/M_GB throughout the length of the MTS. However, at about window position 40, corresponding to approximately MTS position 20000, a positive correlation occurs. This continues until position 50 (MTS position 25000) where the fluctuating returns until the end of the series. This closely follows the segmentation found using HCHC. The same applies for the other two parent variables

with SOT also showing positive and negative correlations and T6T showing varying amounts of positive correlation. The segmentation appears to successfully separate out each of these regions. There are also, however, segmentations that are discovered which do not tie in with the pair-wise relationships apparent in the DCCF. These could be due to more complex relationships not detected in the DCCF.

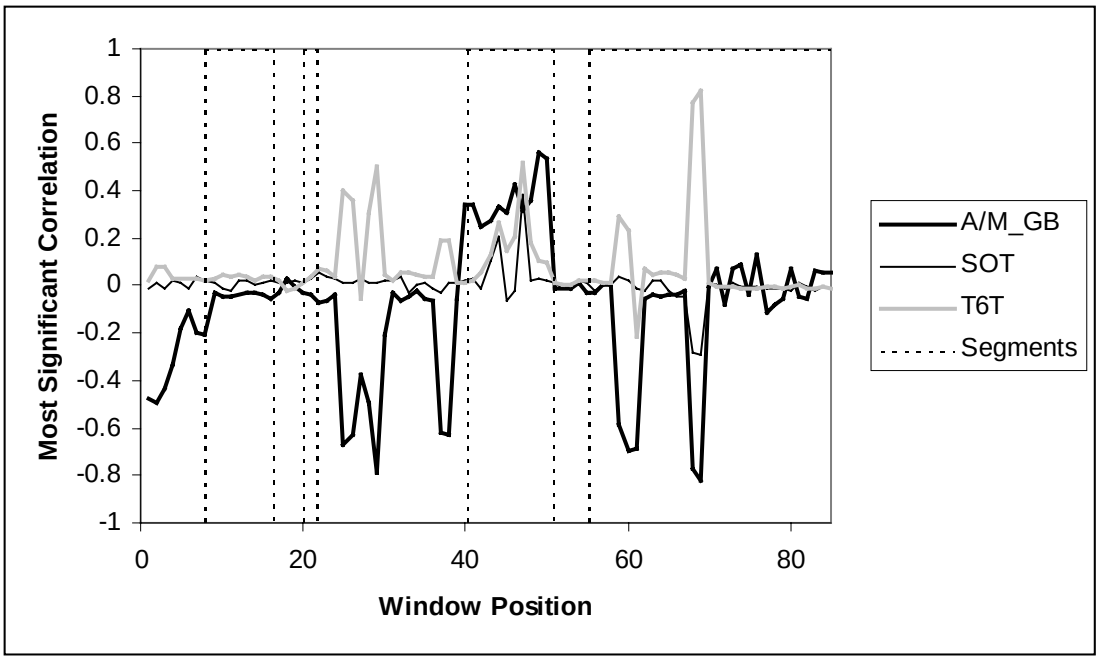


Figure 6.8. Most Significant Correlations for each window position of a DCCF corresponding to TGF and its discovered parents (win_{len} = 1000, win_{jump} = 500). For the Full DCCFs see Appendix F.

In Figure 6.9, the most significant correlation graphs are shown for the variable, BPF. There appears to be far fewer significant correlations between the discovered variables, with sudden bursts of strong correlations, both negative and positive. Upon inspection of the MTS variable, BPF, it was discovered that this variable contained a lot of noise which could have resulted in lower correlations. Despite this, it can be seen once again that, where many of the correlations change, segmentations have been found. For example, the graph representing the

most significant correlation between BPF and RBT shows almost zero correlation until window position 23 where there is a brief positive correlation followed by a brief negative correlation (a segmentation has been discovered between these two regions).

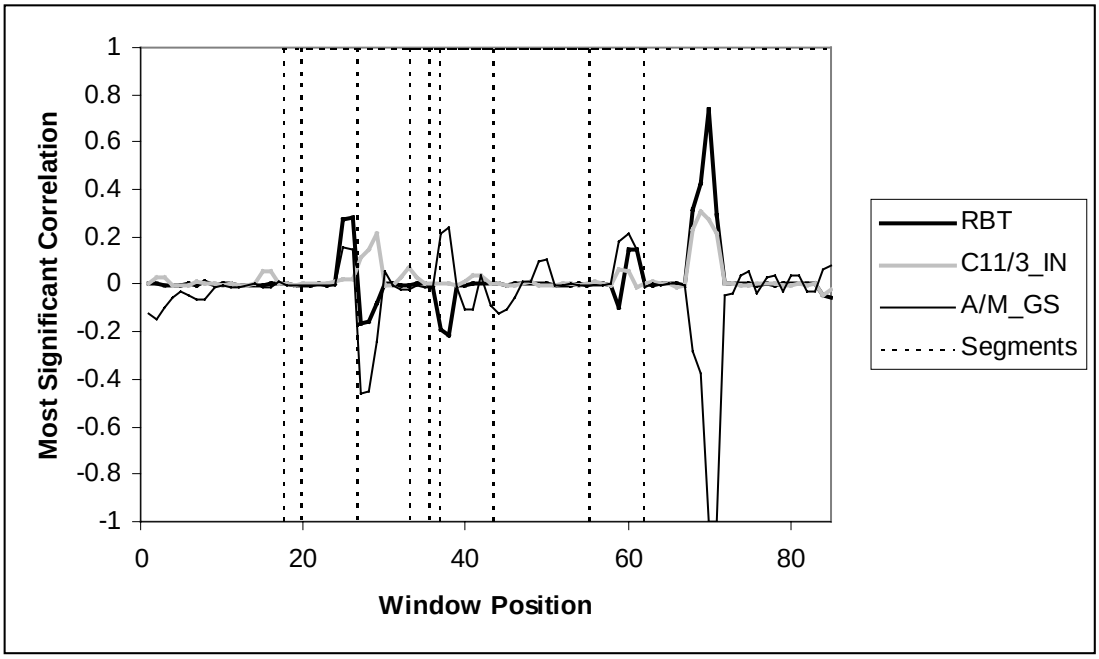


Figure 6.9. Most Significant Correlations for each window position of a DCCF corresponding to BPF and its discovered parents (win_{len} = 1000, win_{jump} = 500). Full DCCFs in Appendix F.

Figure 6.10 shows the results for the variable T36T and similar results are found. In Figure 6.10 T36T and A/M_GB are either negatively correlated or not significantly correlated in the first 17 window positions. At position 18 there is a sudden change to positive correlations and a segmentation has been placed here.

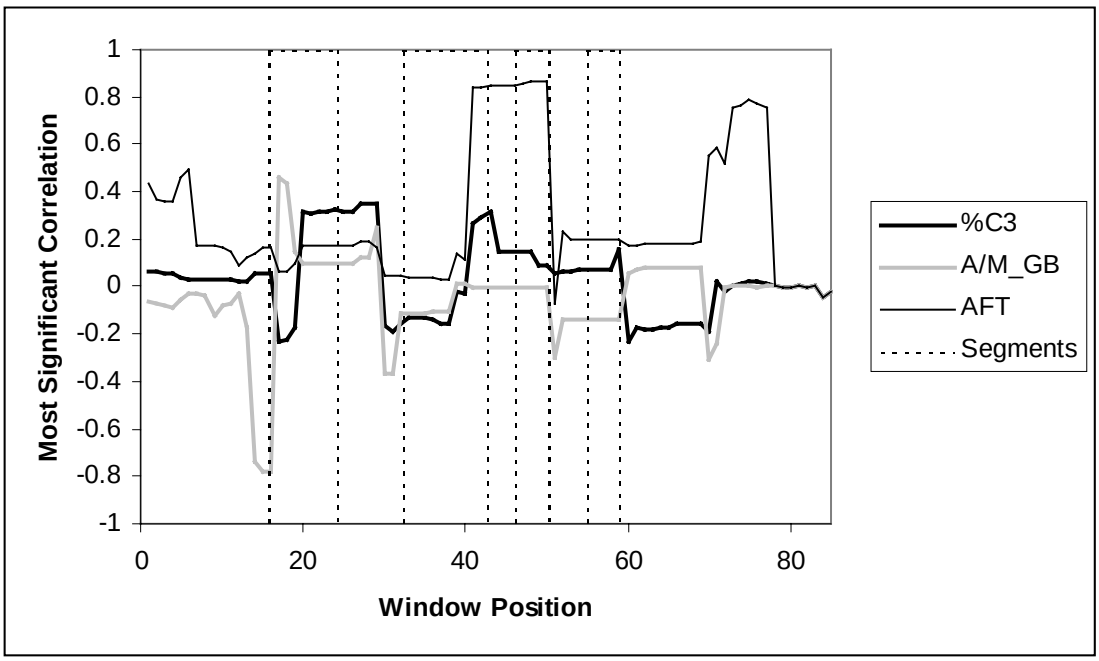


Figure 6.10. Most Significant Correlations for each window position of a DCCF corresponding to T36T and its discovered parents (win_{len} = 1000, win_{jump} = 500). Full DCCFs in Appendix F.

It is interesting to note that most of the segmentations that have been discovered occur where there are switches from positive to negative correlation rather than between regions of strong and weak correlation. It should also be noted that some of the results in Figures 6.7 - 6.10 appear to find a number of segmentations that do not tie in with the correlation changes. This is likely to be because the relationships that are changing are more complex than the pair-wise relationships that are identified in these DCCF analyses and in the next section more complex relationships that are modelled by the DBNs are highlighted.

The complete DBN structure that was discovered for the 21 variable oil refinery MTS can be seen in Figure 6.11. Note how the links are spread relatively evenly over the variables and

lags. Some links span the entire network lag space whilst some have relatively small lag. Most variables bear some influence over another with the exception of about six which appear to be influenced by others whilst having have no effect upon others.

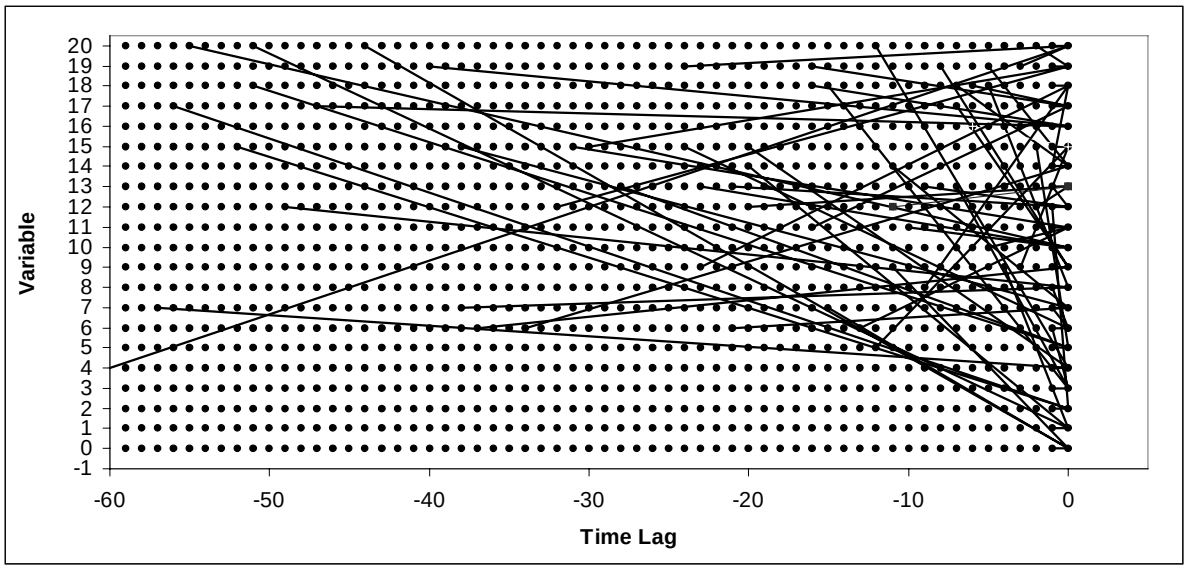


Figure 6.11. The Final DBN Structure discovered using HCHC applied to One Months Oil Refinery Data from 21 Variables. *OpState* nodes are not included.

In general, the results from the process data have been very encouraging with many of the obvious relationships picked up by the DCCF analysis being discovered. It must be noted that other more complex relationships may also have been discovered and the next section looks at some sample explanations that have been generated from the DBNs with *OpState* nodes included to identify these relationships.

6.4 Explanations incorporating Hidden Controllers

Given the discovered structure in Figure 6.11 and parameters for each node in the network including the set of *OpStates*, inference can be applied to generate explanations. This

involves the same process as used in Chapters 3 and 6 whereby certain observations are made about variables in the DBN and inference is used to generate posterior probability distributions over the unobserved variables.

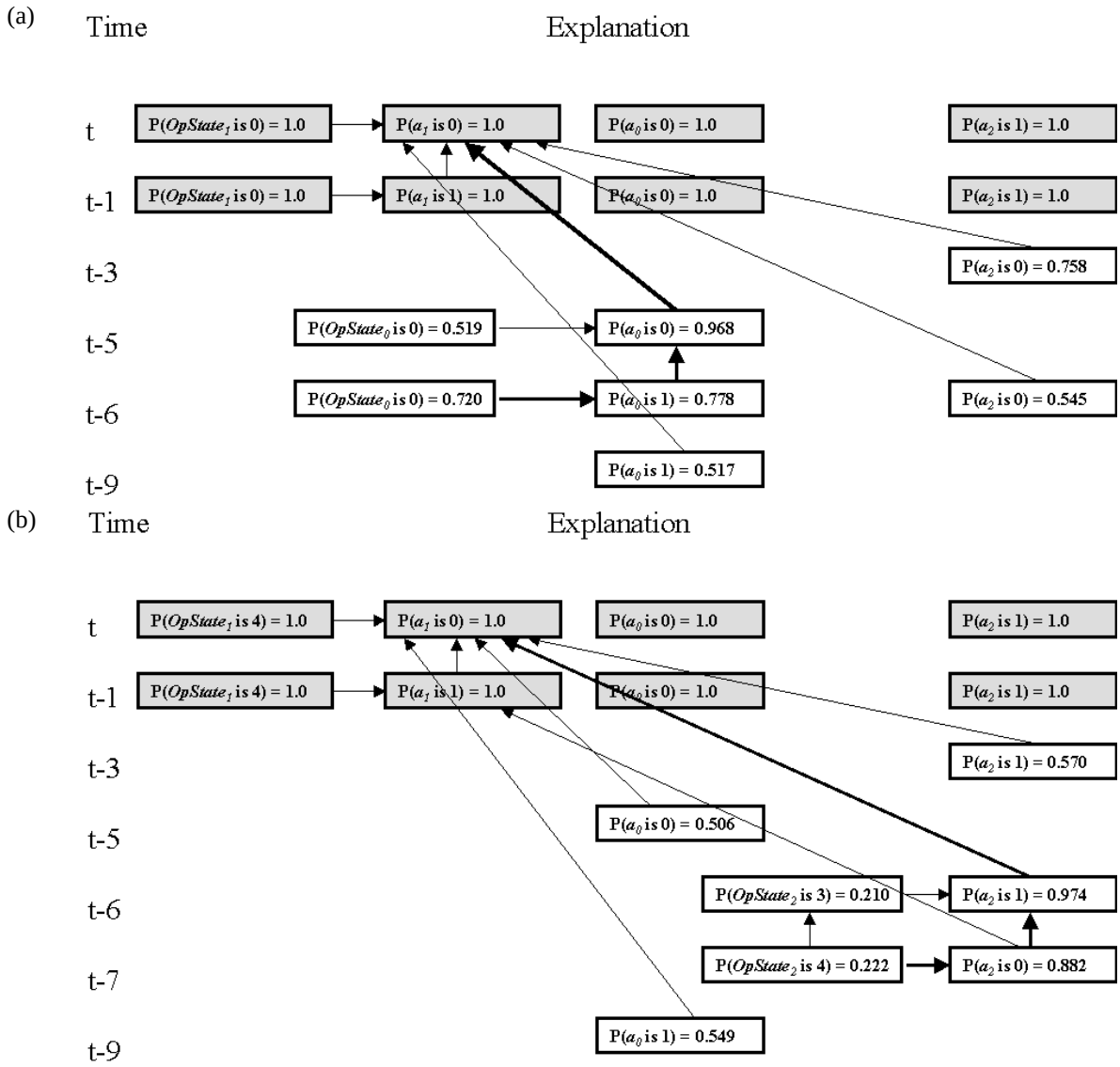


Figure 6.12. Sample Explanations from MTS 3 incorporating Hidden Controllers (*OpStates*).

The explanations are also able to include the *OpStates* as part of the explanation. For example, applying inference to four possible sets of observations on MTS 3 has generated the

explanations in Figure 6.12. In the two explanations, (a) and (b), some observations about variables a_0, a_1 and a_2 are observed as well as the current state of $OpState_1$. It can be seen that the influences other variables have over a_1 vary depending on the state of $OpState_1$. When $OpState_1$ is in state 0, it appears that a_0 being in state 0 with a time lag of 5, has the most effect on variable a_1 but when in state 4, a_2 being in state 1 with a time lag of 6, has the most influence, as highlighted in Figure 6.12 with bold lines.

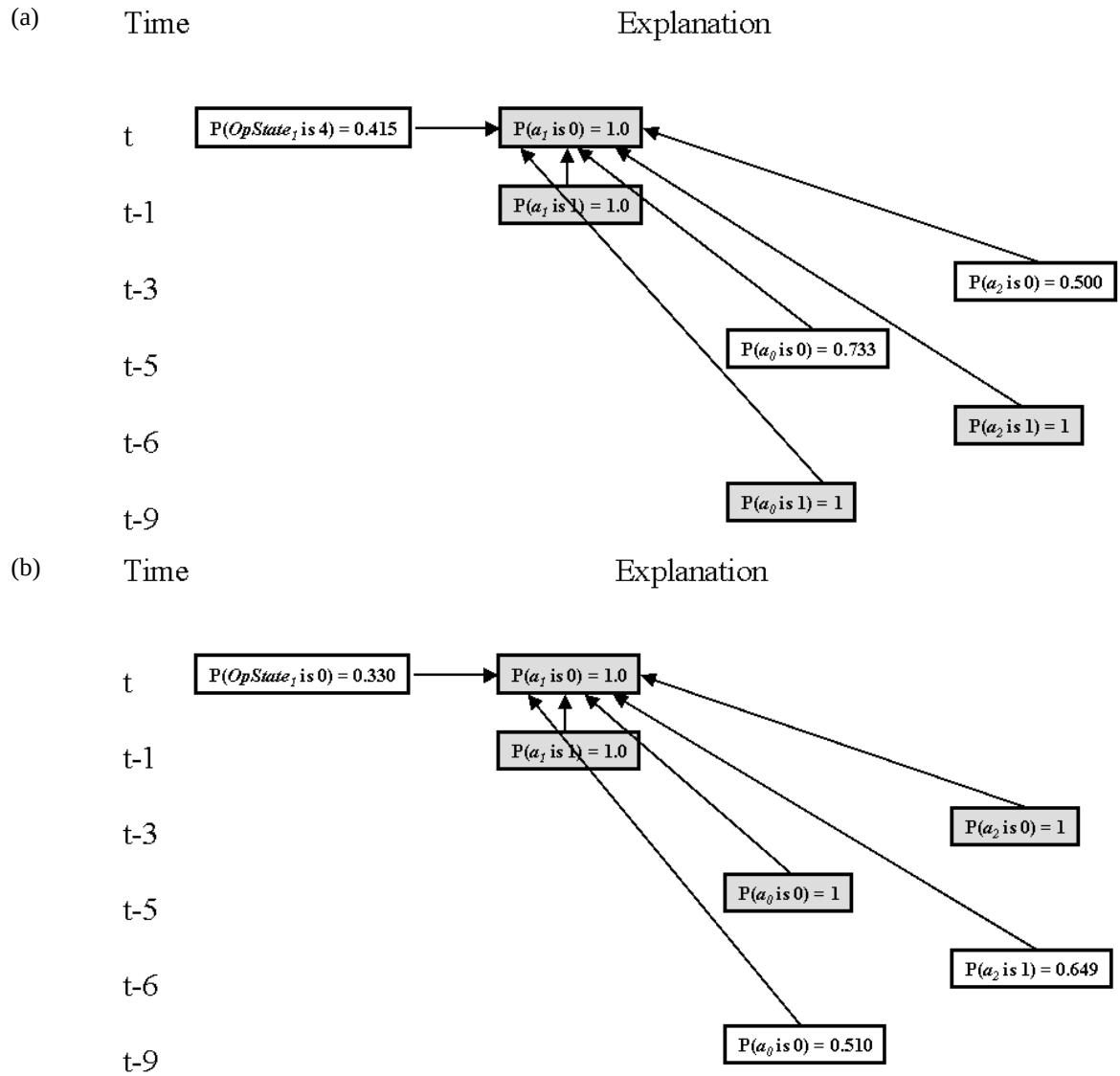
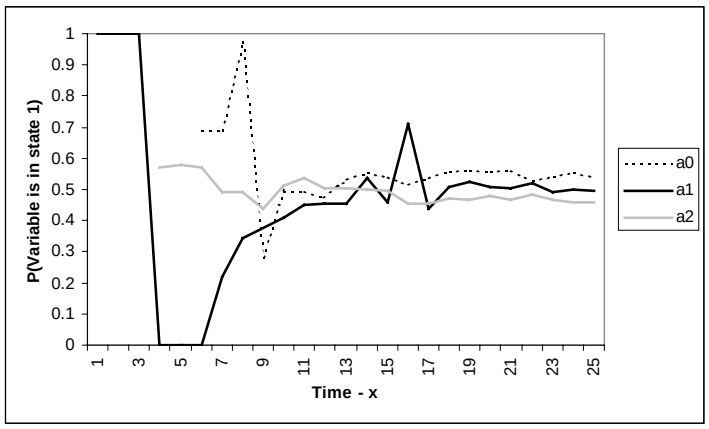


Figure 6.13. Sample Explanations from MTS3 incorporating Hidden Controllers (*OpStates*).

Explanations can also be generated where a change in an *OpState* variable can be the most likely reason for an observation. For example, in 6.13(a) and (b), nothing is observed about any *OpState* variables. However, given the set of observations, the most probable state for $OpState_1$ can be inferred. The posterior distribution over time are displayed for explanation 6.12(a) and 6.12(b) in Figure 6.14, below.

(a)



(b)

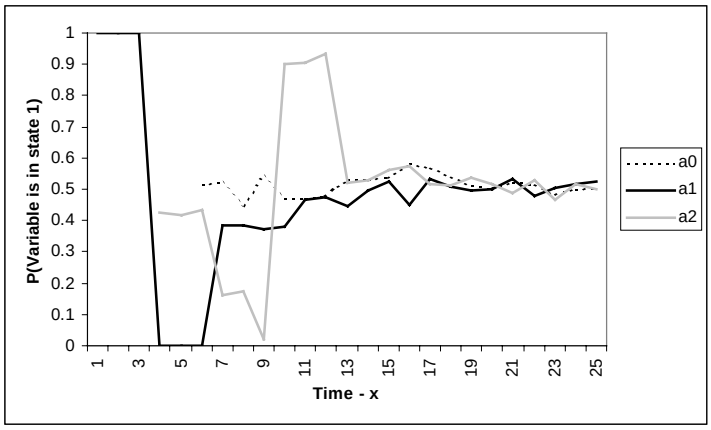


Figure 6.14. The posterior probabilities of three variables in MTS 3 as inference is applied back in time given differing values of *OpStates*. (a) All *OpStates* = 0, (b) All *OpStates* = 4. Notice how the *OpStates* have affected the probabilities over the variables. a_0 is the most likely reason for a_1 changing from state 1 to state 0 when *OpStates* = 0 (having a positive effect with a lag of 5) but a_2 is the most likely reason if *OpStates* = 4 (having a negative effect with a lag of 6).

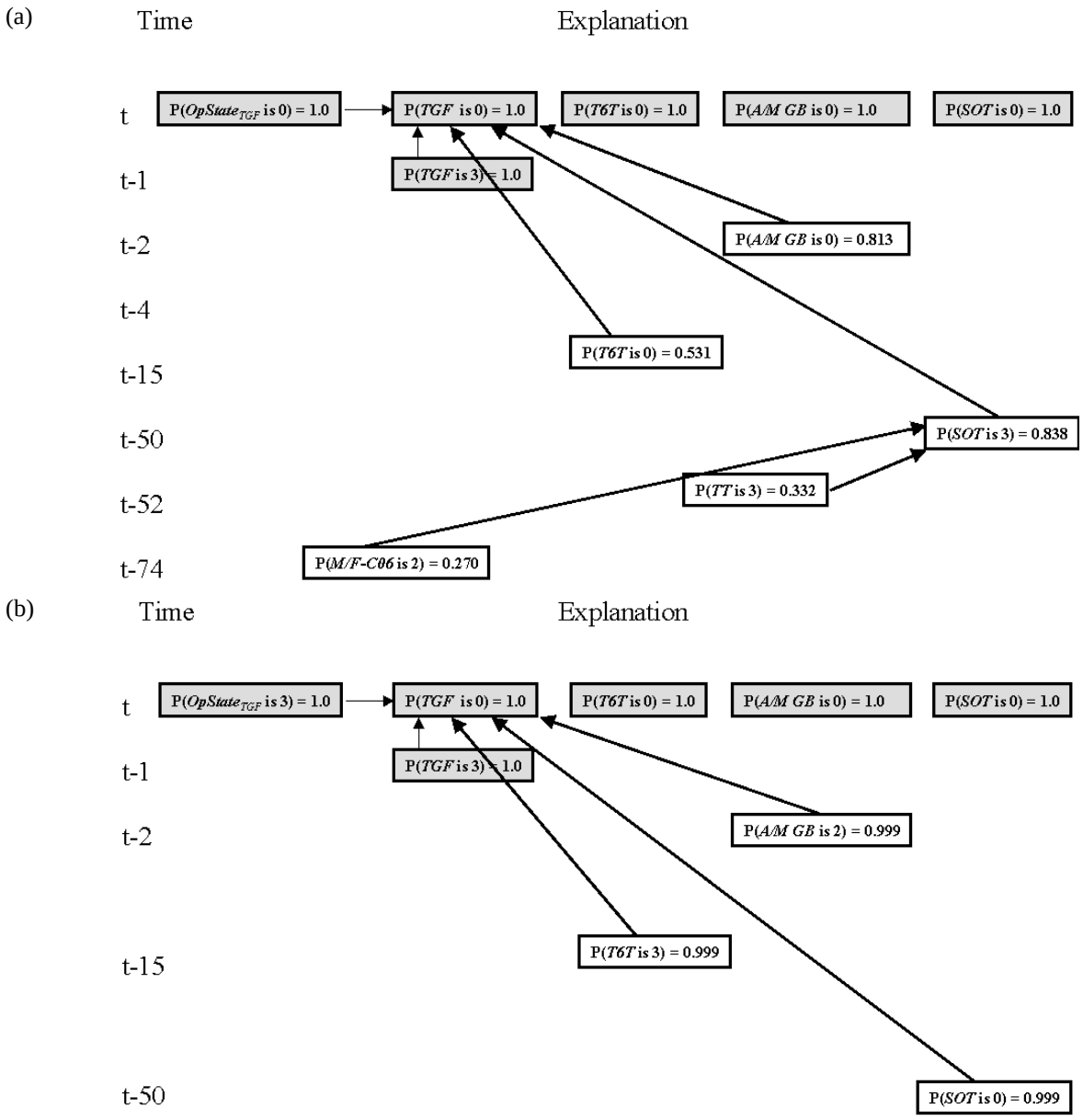


Fig 6.15. Sample of Generated Explanations from the Oil Refinery DBN.

Figure 6.15 and 6.16 show some more explanations that have been generated but using the DBN discovered from the oil refinery data. It can be seen in 6.15(a) that SOT has a strong likelihood of being in state 3 given the instantiations and $OpState_{TGF}$ being 0. The posterior probabilities change drastically when $OpState_{TGF}$ changes to 3 as is shown in 6.15(b) with the

most probable state for each parent variable altering (SOT now most likely to have been in state 3 and A/M_GB in state 2). In 6.16(a) the effect of adding some new evidence to 6.15(b) is shown which changes all the variable states again (if C11/3 is known to have been in state 3, 16 minutes ago, SOT will now most likely have been in state 3 and A/M_GB in state 0). In 6.16(b) TGF and its set of parents are instantiated resulting in the controller variable $OpState_{TGF}$ being most likely in state 0.

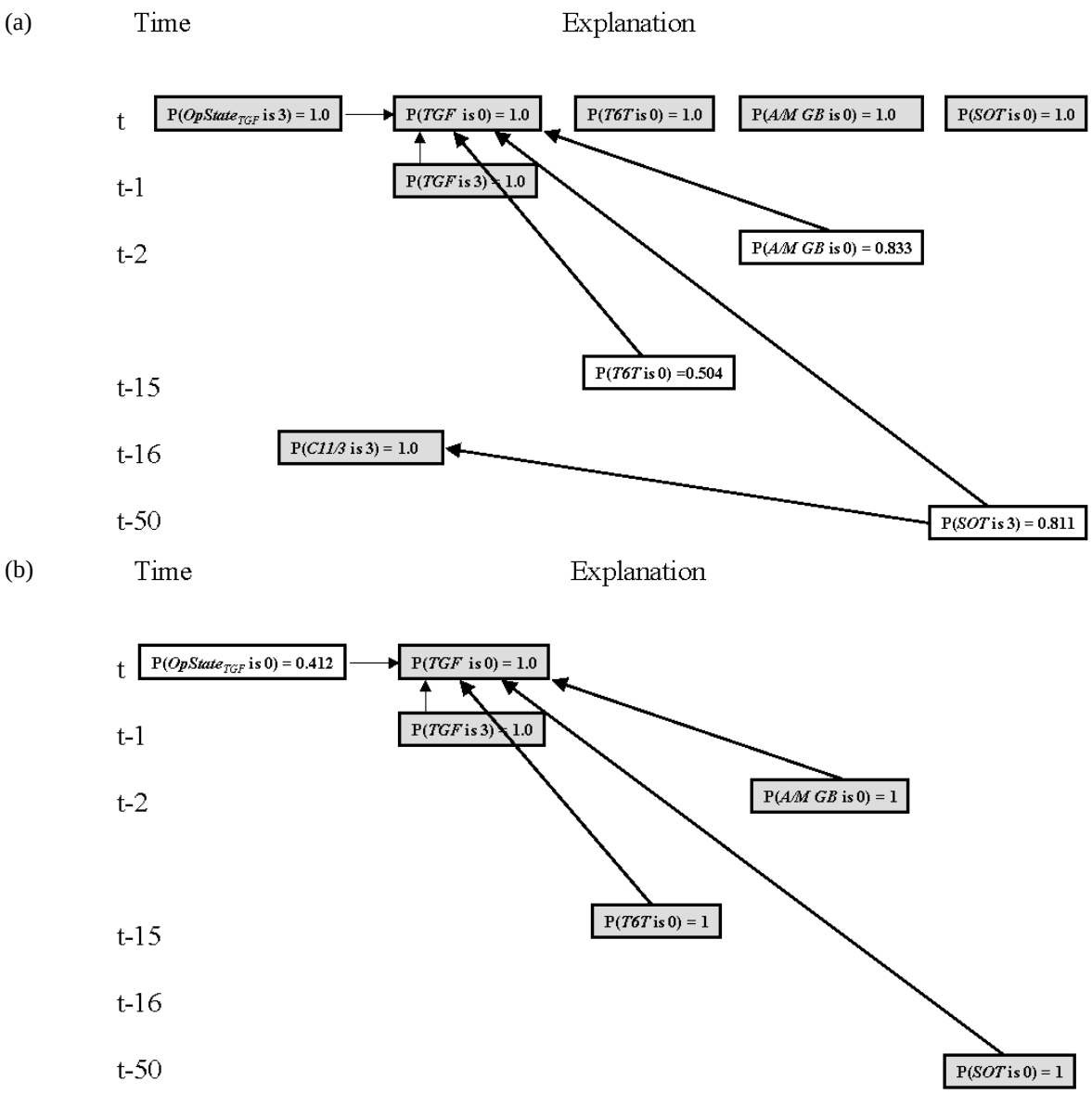


Fig 6.16. Sample of Generated Explanations from the Oil Refinery DBN.

In addition, by changing the value of A/M-GB it was observed that TGF was tightly coupled with it. In other words, when A/M-GB was high then TGF was high and low when TGF was low. However, when T6T was high and SOT was low, it appeared that the relationship between A/M-GB and TGF disappeared (TGF was generally in state 1 when A/M-GB was 3). The most likely state of $OpState_{TGF}$ was seen to change, implying that an alternative operating state occurred when these configurations were encountered. This illustrates how the segmentation of the MTS based on the discovered links includes more complex relationships than the simple pair-wise ones shown in the DCCF cross-sections of Figures 6.8-6.10.

6.5 Conclusions

Experiments in this chapter indicate that hidden nodes such as *OpState* can be used to model changing dependencies within a MTS. What is more they offer a method for doing so whilst remaining transparent. In other words, the node can easily be interpreted as 'the current state of the system' and, therefore, be used within explanations of events.

The hill climb procedure which reduces the search space drastically through the use of a specific representation has allowed good models to be learnt from the oil refinery MTS. Many of the segmentations discovered, tie in with the pair-wise correlations found in the DCCF but some were found that do not, most likely due to the relationships being more complex. The generation of explanations given certain test observations shows that the paradigm is capable of modelling more complex relationships between groups of variables that go beyond simple pair-wise relationships. For example, a pair-wise correlation may exist between two variables but only whilst another variable or set of variables are in a particular

configuration. This is often evident in data such as from a chemical process, where two temperatures may be closely coupled, but if another flow rate increases or another temperature increases, the effect may be lost between them.

7 Discussion

This chapter discusses conclusions and possible future directions for the research covered in this thesis. Section 7.1 is concerned with drawing conclusions from the results documented within the previous chapters, including the pros and cons of the methods explored. Section 7.2 deals with ways in which the methods can be expanded and improved upon.

7.1 Conclusions

Within this thesis, a framework has been set out for *automatically* explaining *high-dimensional*, long Multivariate Time Series (MTS) with *large possible time lags* and with *little time*. This framework consists of solving two problems: Firstly, the explanation of new data as quickly as possible using Dynamic Bayesian Networks (DBNs) and secondly the off-line generation of these models from MTS with changing dependencies. Both involve pre-processing such MTS in order to reduce their dimensionality based on pairwise correlation between variables. A method has been proposed that is capable of doing this where a direct trade-off can be made on accuracy and speed and has been shown on synthetic and real data to 'group' MTS variables into related subsets very efficiently, even when time is very short. Having broken down the original MTS into several smaller-dimensional MTS, the search spaces involved in learning probabilistic models for explanation are still massive and an approximate method has been investigated in this thesis which manages to find good models very efficiently when compared to other standard techniques. Extending these sorts of models for MTS with dependencies that can change between variables has also been investigated

with success in keeping the models transparent and the explanations easy to interpret by a layperson. The contribution to knowledge can be summarised as follows:

1. A new method for grouping MTS variables where dimensionality is very high, time lag can be large and/or time is short. This can be scaled to very large MTS in order to reduce the dimensionality of model building.
2. A comparison of standard static BN search methods that have been adapted to search for DBN structure
3. A new method has been introduced for efficiently searching for DBN structure known as EP-Seeded GA.
4. A method for incorporating changing dependencies into the representation whilst ensuring transparency has been achieved with some success by using hidden discrete variables to represent the changing states of dependencies.
5. The above methods have allowed the rapid automatic explanation of synthetically generated data and chemical process data through using a combination of the above methods and transparency has been ensured throughout so that at the end of each process (e.g. grouping, model building etc.) a non-statistical user can intervene and adjust the models.

In general the presented framework has been very successful in solving the problems set out in Chapter 1. It is fully automated. No user intervention is required in the entire process from MTS as input to explanation model at the other end, once parameters are set. Of course, if need be, the user can examine the end result of each stage such as the groupings of the MTS variables, the correlations that generated these groupings and the resulting DBN models.

They can even update them based on background knowledge due to the transparency of each stage. The generated explanations are easy to interpret, consisting of simple probabilistic dependencies between variables due to the transparency of the DBN paradigm. The evolutionary approach adopted, along with the heuristics used, have shown to be efficient in comparison to standard methods that have been adapted to DBN learning. What is more the quality of the resultant models were very good with respect to structural difference on synthetic data and the oil refinery data generated good models that were parsimonious with the expectations of experts and process diagrams.

The pre-processing grouping of MTS variables can be achieved very quickly making it ideal for pre-processing in large, time constrained problems. This is achieved by using approximate techniques for both the correlation search and the grouping of variables. What is more, by altering the parameters, more precise groupings can be found given more time. In other words, unlike other deterministic clustering methods, the method can be tailored to suit the application - if time is not an issue then an exhaustive search can be carried out over all correlations but if time is limited the extensiveness of the search can be controlled by altering the parameters. The genetic grouping algorithm also holds the advantage over other deterministic clustering methods that it will not be affected by local maxima due to the nature of genetic algorithm search. This pre-processing method for large dimensional time series has been successfully employed in this thesis to break down chemical process data rapidly before applying model building.

The EP-Seeded GA makes good use of approximate techniques to increase the efficiency of learning DBN with large time lags by utilising a similar list to that sought after in the pre-processing stage. The set of heuristics has assisted in narrowing the search space and speeding convergence. It has compared favourably to other standard search methods for Bayesian networks, which have been adapted to search for dynamic Bayesian networks. However, it is likely that there are other ways to improve the algorithm further in order to improve efficiency further such as making the parameters to the seeded GA dynamic. For example, the crossover rate could be lowered and the mutation rate raised, as the number of generations increases. This will mean that as the algorithm proceeds as existing triples are exhausted in the number of combinations, new triples will be introduced more and more to try and improve the final fittest chromosome.

In terms of learning models with changing dependencies, the results have been promising on both synthetic and real data in terms of good explanations. The hidden controller nodes are treated like any other node in the resultant models and are, therefore, easily interpreted. For example, an explanation for variable 1 in an MTS may be 'because $OpState_1$ changed from state 0 to state 1, five minutes ago' where $OpState_1$ represents some hidden controller for variable 1. On the oil refinery data, some interesting relationships have been discovered using the changing dependency paradigm. For example, when the state of hidden controller for a particular flow rate changed, the relationship with each of its parent variables (all temperatures) was noticed to alter. This change can sometimes be quite dramatic and this behaviour was sometimes observed between variables where a strong negative relationship became a strong positive one.

Several problems have been encountered through the investigations carried out within this thesis. The most prominent problem, which applies to several stages of the methodology, is the selection of parameters. Both the pre-processing grouping algorithm and the EP-Seeded GA are heavily parameterised. The grouping algorithm relies heavily on the size of the correlation list, as this will determine how prospective groupings are scored. It is essential to get this correct and yet very difficult to determine. Fortunately, for the oil refinery application, the results were very good due to the even spread of high correlations and the groupings made good intuitive sense. However, there may be some applications where correlations between a few variables may dominate the list and, therefore, displace relatively good correlations from the list. In these situations, the size of the list must be carefully determined. Other very important parameters will be the population sizes, the number of generations, and the mutation and crossover rates in the Evolutionary Program and Genetic Algorithm of the EP-Seeded GA. These will all have a large effect on the efficiency of the search and the quality of the final model.

Another possible problem that may be encountered is due to discretisation. Discretisation can bias the model learning phase dramatically. The method adopted within this thesis, i.e. frequency-based discretisation, was chosen as it relatively simple, quick, and maximises the frequency of each state. However, like all discretisation methods, information is lost when applied and so this must be minimised, particularly when data is not very rich. Explanations may also be misleading when talking about states of a discretised variable. For example, frequency-based disretisation may result in three states that have very close bounds and one state which covers most of the variables range depending on how frequent each of these

states are observed within the data. Therefore, some states which may be assumed as relatively low are in fact in the higher ranges of possible value. However, as long as the bounds of each variable's state are made clear, misinterpretation should be minimised.

Finally, the learning of models with changing dependencies that generated some interesting results when using a simple Hill Climb search was extremely slow. The Structural EM algorithm was hoped to be a more effective and speedy method than the hill climb. However, the results proved otherwise, suffering from local maxima and the calculation of expected statistics taking a long time. However, it is hoped that future research will involve using standard methods such as deterministic annealing to overcome local maxima. The next section discusses some of the future research, which is planned to overcome some of the problems that were encountered and suggests ways of extending the framework.

7.2 Further Research Directions

Combining the entire procedure into one continuous process is to be investigated including the effect of different quality outputs from one process (e.g. grouping) being used as input to the next (e.g. EP-Seeded GA). Due to the input/output of each stage in the methodology being transparent, it has been possible to assess the quality of each process individually. This is useful in allowing a user to correct any obvious errors and add user knowledge. However, it will be interesting to see how the effects of one bad process propagate through to the final explanations. A more challenging task may be to combine all of these processes into one iterative algorithm which tries to improve the outputs of each stage given the outputs of the

other stages. Therefore, the results from the grouping stage will not necessarily be left unchanged during the EP-Seeded GA stage.

A 61 variable MTS of length 1000 has been grouped within 5 minutes and 58 seconds on a standard Pentium PC. A group with 20 variables has been selected and a DBN learnt using EP-Seeded GA taking 3 minutes and 29 seconds. Explanation has been generated using this DBN in 4 minutes and 3 seconds. Expanding on these initial timing experiments, it will be useful to see how the algorithms can be optimised to speed the explanation generation further. If the methodology is to be utilised in real-world applications, it is the actual time in minutes and seconds that will be important and this is to be investigated further.

The problem identified in the previous section concerning parameter estimation for the grouping algorithm is to be explored in greater detail. A method for determining the size of the list by imposing a distribution over correlations (all variables and lags) is currently underway. It is known that the correlations in a bivariate time series process can be approximated by a normal distribution [Fisher15, Lush31]. However, it is not known whether this extends to MTS data. If a distribution can be found for the correlations of a MTS, this will allow the size of the list to be calculated with a certain confidence (e.g. 95%) based upon the number of variables and maximum time lag.

The calculation of pairwise correlation is carried out twice within the framework described in this thesis. Firstly, a statistical correlation coefficient is used on the data to search for a list of highly correlated variable pairs as part of the grouping algorithm. A similar procedure is carried out on the pre-processed data for the EP part of EP-Seeded GA to speed convergence,

where a Bayesian network scoring metric is used to score single links. It would be useful if these two searches could be combined in some way to speed up the overall efficiency of the methodology. It must be noted that these searches do not perform precisely the same function and, therefore, each search has different valid pairings. For example, the groupings algorithm only requires storing a link from a_i to a_j with one possible lag. This is because once a good correlation is found between two variables, the lag is unimportant. In contrast, the EP for EP-Seeded GA requires storing pairs of variables with various different lags because there may be several good links between two variables with different time lags. Some process could be carried out which would combine the two present searches, as there will be substantial overlap between the two.

As mentioned in the previous section, a good discretisation policy is important in generating good explanations. It is intended that many more MTS discretisation policies will be looked at, particularly in how they affect the final explanations in a model. Methods exist for learning such policies whilst learning network structure simultaneously. However, in time-restricted situations the time required to learn policies is likely to be a problem and a simpler quicker method will be preferred. Another way to avoid the problems associated with discretisation would be to look at extending continuous BNs into the temporal domain. Continuous BNs, [Hofmann96, John95], model continuous data using dependency structure and probability density functions between groups of variables. There has been some work on learning these from data, usually with Gaussian distribution assumptions [Geiger94]. Modelling MTS, such as the oil refinery data, in this way allows explanations to involve more complex density functions rather than simpler distributions over states of variables.

What is more, the problem of storing larger and larger conditional probability tables as the number of links increase is avoided.

Confidence in learnt networks is vital if explanations are to be used in real world scenarios. For this reason, methods to compute statistics which reflect the confidence in an explanation must be investigated. This research has begun to some degree using bootstrapping methods [Efron93]. Friedman et al. have used bootstrapping to calculate the confidence that data supports a particular feature such as a set of links in a BN, [Friedman99]. It would be invaluable if these confidence statistics could be incorporated into explanation such as those found in Chapters 5 and 6.

Another important direction for this research is to experiment with as many different types of MTS data as possible. Whilst some of the algorithms introduced in this thesis have been tested on synthetic data, oil refinery data and also some on visual field data, it is still essential to see how they, and the other algorithms, will behave on many other varied datasets. It is planned to test the methodology on various datasets including EEG data, gene expression data and robot sensor data, in order to identify the common characteristics and differences between certain MTS. This will allow the algorithms to be generalised or specialised to particular applications. For example it has already been discovered that the grouping algorithm is more sensitive to datasets where the correlations are less evenly spread such as visual field data as opposed to the oil refinery data where many variables are strongly correlated with an even spread.

The process of learning models with changing dependencies from MTS may be speeded up in various ways. The techniques used in Chapter 5 could be used, such as the EP-Seeded GA type search for structure, whilst a hill climb or some heuristic driven EM algorithm could be applied to the parameters for the hidden variables. Some classic problems were encountered with SEM and the investigation as to whether methods such as deterministic annealing [Ueda95] can be applied to overcome these will be explored. Another problem that may be encountered with some MTS is the over-connectivity of these networks with hidden operating variables. If a variable has many operating states then it is likely that it will require many parents for each state. A paradigm known as MultiNets [Bilmes2000] may be used to explore ways to avoid this where links are removed based on the states of the hidden variables to simplify the structure and speed inference.

References

[Baeck93] T. Baeck, G. Rudolph and H.-P. Schwefel, “Evolutionary Programming and Evolution Strategies: Similarities and Differences”, D.B. Fogel and W. Atmar, editor: Proceedings of the 2nd Annual Conference on Evolutionary Programming, 11-22, 1993.

[Baeck96] T. Baeck, “Evolutionary Algorithms: Theory and Practice”, Oxford University Press, 1996.

[Bakshi94] Bakshi, Stephanopoulos, “Representation of Process Trends – III. Multiscale Abstraction of Trends from Process Data”, Comp Chem Eng Vol. 18 No. 4, pp. 267-302, 1994.

[Berndt96] D. Berndt, J. Clifford, “Finding Patterns in Time Series: A Dynamic Programming Approach”, Advances in Knowledge Discovery and Data Mining, AAAI Press, MIT Press, 1996.

[Bilmes2000] J. A. Bilmes, “Dynamic Bayesian Multinets”, Proceedings of the 16th Annual Conference on Uncertainty in AI, pp. XX-XX, 2000.

[Binder97] J. Binder, K. Murphy, S. Russell, “Space-Efficient Inference in Dynamic Probabilistic Networks”, Proceedings of the 15th International Joint Conference on Artificial Intelligence, pp. 1292-1296, 1997.

[Bouckaert94] R. R. Bouckaert, “Probabilistic Network Construction Using the Minimum Description Length Principle”, Technical Report RUU-CS-94-27, Dept of Computer Science, Utrecht University, July 1994.

[Box76] G. E. P. Box, G. M. Jenkins, “Time Series Analysis, Forecasting and Control”, 2nd edition, Holden-Day, San Francisco, 1976.

[Boyen98] X. Boyen, D. Koller, “Tractable Inference for Complex Stochastic Processes”, Proceedings of the 14th Annual Conference on Uncertainty in AI, pp. 33-42, 1998.

[Buntine96] W. Buntine, “A Guide to the Literature on Learning Probabilistic Networks form Data”, IEEE Transactions on Knowledge and Data Engineering 8, no. 2, pp. 195-210, 1996.

[Chang94] I.-C. Chang, C. Yu, C. Liou, “Model-Based Approach for Fault Diagnosis. 1. Principles of Deep Model Algorithm”, Ind. Eng. Chem. Res., 33, pp. 1542-1555, 1994.

[Chatfield89] C. Chatfield, “The Analysis of Time Series - An Introduction”, Chapman and Hall, 4th edition, 1989.

[Chickering96a] D. M. Chickering. “Learning Bayesian Networks is NP-Complete”, AI and Statistics, pp. 121-30, 1996.

[Chickering96b] D.M. Chickering, D. Heckerman “Efficient Approximations for the Marginal Likelihood of Bayesian Networks with Hidden Variables”, Technical Report MSR-TR-96-08, Microsoft Research, 1996.

[Chow68] C. K. Chow, C. N. Liu, “Approximating Discrete Probability Distributions with Dependence Trees”, IEEE Transactions on Information Theory, Vol. 14, pp. 462-467, 1968.

[Cooper90] G.F. Cooper, “The Computational Complexity of Probabilistic Inference

using bayesian Belief Networks", Artificial Intelligence 42, pp. 393-405, 1990.

[Cooper92] G.F. Cooper, E. Herskovitz, "A Bayesian Method for the Induction of Probabilistic Networks from Data", Machine Learning, Vol. 9, pp. 309-347, 1992.

[Cooper93] G.F. Cooper, "A Method for Learning: Belief Networks that contain Hidden Variables", Proceedings of the Workshop on Knowledge Discovery in Databases, pp. 112-124, 1993.

[Dagum92] P. Dagum, A. Galper, E. Horvitz, "Dynamic Network Models for Forecasting", Proceedings of the 8th Annual Conference on Uncertainty in AI, pp. 41-48, 1992.

[Dagum95] P. Dagum, A. Galper, E. Horvitz, A. Seiver, "Uncertain Reasoning and Forecasting", International Journal of Forecasting 11, pp. 73-87, 1995.

[DeKleer91] J. DeKleer, B. Williams (Eds), "Qualitative Reasoning about Physical Systems", Artificial Intelligence 51, Numbers 1-3, 1991.

[Dempster76] A. P. Dempster, N. M. Laird, D. B. Rubin, , "Maximum Likelihood from Incomplete Data", Journal of the Royal Statistical Society, pp. 1-38, 1976.

[Dhurjati87] P.S. Dhurjati, D.E. Lamb, D.L. Chester, "Experience in the Development of an Expert System for Fault Diagnosis in a Commercial-Scale Chemical Process", FOCAPO-87, CACHE/Elsevier, 589, 1987.

[Efron93] B. Efron, R.J. Tibshirani, "An Introduction to the Bootstrap", Chapman & Hall, London, 1993.

[Falkenauer98] E. Falkenauer, “Genetic Algorithms and Grouping Problems”, Wiley, 1998.

[Fisher15] R. A. Fisher Frequency Distribution of the Values of the Correlation Coefficient in Samples from an Indefinitely Large Population, Biometrika, Vol. 10, No. 4, pp. 507-521, 1915.

[Fisher21] R. A. Fisher, “On the 'Probable Error' of a Coefficient of Correlation Deduced from a Small Sample”, Metron, 1, pp. 4-32, 1921.

[Fogel95] D.B. Fogel, “Evolutionary Computation - Toward a New Philosophy of Machine Intelligence”, IEEE Press, 1995.

[Forbes95] J. Forbes, T. Huang, K. Kanazawa, and S. Russell. The BATmobile: Towards a Bayesian Automated Taxi. In Proceedings of the 12th International Joint Conference on Artificial Intelligence, 1995.

[Friedman96] N. Friedman, M. Goldszmidt, “Discretizing Continuous Attributes While Learning Bayesian Networks”, Proceedings of the 13th International Conference on Machine Learning, pp. 157-165, 1996.

[Friedman97] N. Friedman, “Learning Belief Networks in the Prescence of Missing Values and Hidden Variables”, International Conference on Machine Learning, 1997.

[Friedman98a] N. Friedman, “Learning the Structure of Dynamic Probabilistic Networks”, Proceedings of the 14th Annual Conference on Uncertainty in AI, pp. 139-147, 1998.

[Friedman98b] N. Friedman, “The Bayesian Structural EM Algorithm”, Proceedings of

the 14th Annual Conference on Uncertainty in AI, pp. 129-138, 1998.

[Friedman99] N. Friedman, M. Goldszmidt, A. Wyner, "Data analysis with Bayesian networks: A bootstrap approach", Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence, pp. 206-215, 1999.

[Garey79] M. Garey and D. Johnson, "Computers and Intractability - A Guide to the Theory of NP-Completeness", W.H. Freeman, San Francisco, 1979.

[Geiger94] D. Geiger, D. Heckerman, "Learning Gaussian Networks", Technical Report, MSR-TR-94-10, Microsoft Research, 1994.

[Gharamani98] Z. Ghahramani, "Learning Dynamic Bayesian Networks, Adaptive Processing of Sequences & Data Structures", Lecture Notes in AI, Springer, pp 168-197, 1998.

[Gharamani99] Z. Ghahramani, G. E. Hinton, "Variational Learning for Switching State-Space Models", Neural Computation, Vol. 12, No. 4, pp. 963-996, 1999.

[Goldberg85] D.E. Goldberg and R. Lingle, "Alleles, Loci, and The Travelling Salesman Problem", Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp. 154-159, 1985.

[Goldberg89] D. E. Goldberg, "Genetic Algorithms in Search, Optimisation, and Machine Learning", Addison Wesley, 1989.

[Gordon88] K. Gordon, A. F. M. Smith, "Modelling and Monitoring Discontinuous Changes in Time Series", Bayesian Analysis of Time Series and Dynamic Linear Models, New York, Marcel Decker, pp. 359-392, 1988.

[Haykin94] S. Haykin, “Neural Networks. A Comprehensive Foundation”, Chapter 13, Macmillan, 1994.

[Heckerman95] D. Heckerman, D. Geiger, D. Chickering, “Learning Bayesian Networks: The Combination of Knowledge and Statistical Data”, Machine Learning Vol. 20, No.3, 1995.

[Heckerman97] D. Heckerman, C. Meek, G. Cooper,, “A Bayesian Approach to Causal Discovery”, Technical report, MSR-TR-97-05, Microsoft Research, 1997.

[Henrion88] M. Henrion, “Propagating uncertainty in Bayesian networks by probabilistic logic sampling ”, Proceedings of the 2nd Annual Conference on Uncertainty in AI, pp. 149-163, 1988.

[Hofmann96] R. Hofmann, V. Tresp, “Discovering Structure in Continuous Variables using Bayesian Networks”, Advances in Neural Information Processing Systems 8, MIT Press, Cambridge MA, 1996.

[Holland95] J. H. Holland, “Adaptation in Natural and Artificial Systems”, University of Michigan Press, 1995.

[John95] G. John, P. Langley, “Estimating Continuous Distributions in Bayesian Classifiers”, Proceedings of the 11th Annual Conference on Uncertainty in AI, pp. 338-345, 1995.

[Kanazawa95] K. Kanazawa, D. Koller, S. Russell, “Stochastic Simulation Algorithms for Dynamic Probabilistic Networks”, Proceedings of the 11th Annual Conference on Uncertainty in AI, pp. 346-351, 1995.

[Kim83] J. H. Kim, J. Pearl, “A Computational Model for Combined Causal and

Diagnostic Reasoning in Inference Systems", Proceedings of the 8th International Joint Conference on AI, pp. 190-193, 1983.

[Kim94] C.-J. Kim, "Dynamic Linear Models with Markov-Switching", Journal of Econometrics 60, pp. 1-22, 1994.

[Knuth69] D. E. Knuth, "Art of Computer Programming, Fundamental Algorithms", Vol. 1, pp. 46, 1969.

[Koller97] D. Koller, D. McAllester, and A. Pfeffer, "Effective Bayesian Inference for Stochastic Programs",. Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 740-747, 1997.

[Koza92] J. Koza, "Genetic Programming: On the Programming of Computers by Natural Selection", MIT Press, 1992.

[Kramer87] M.A. Kramer, "Malfunction Diagnosis Using Quantitative Models with Non Boolean Reasoning in Expert Systems", AIChE Journal, Vol. 33, No. 1, pp. 130-140, 1987.

[Lam94] W. Lam, F. Bachus, "Learning Bayesian Networks: An Approach Based on the MDL Principle", Computational Intelligence, Vol. 10, No. 4, 1994.

[Lang88] K. J. Lang, G. E. Hinton, "The Development of the Time Delay Neural Network for Speech Recognition", Technical Report CMU-CS-88-152, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[Larranaga96] P. Larranaga, M. Poza, Y. Yurramendi, R. Murga, C. Kuijpers, "Structure Learning of Bayesian Networks using GAs", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 18, No.9, pp. 912-926, 1996.

[Lilliefors67] H. W. Lilliefors, “On the Kolmogorov-Smirnov Test for Normality with Mean and Variance Unknown”, Journal of the American Statistical Association, No. 62, pp. 399-402, 1967.

[Lush1931] Lush, J., L., “Predicting Gains in Feeder Cattle and Pigs”, Journal of Agricultural Research, 42, pp. 853-881, 1931

[Lutkepojl93] H. Lutkepojl, “Introduction Multiple Time Series Analysis”, Springer-Verlag, 1993.

[McDermott84] J. McDermott, “R1 Re-Visited: 4 Years in the Trenches”, The Artificial Intelligence Magazine, pp. 21-32, 1984.

[Meila97] M. Meila, M. I. Jordan, Q. Morris, “Estimating Dependency Structure as a Hidden Variable”, Neural Information Processing Systems 10, NIPS97, MIT Press, 1997.

[Mirkin99] B. Mirkin, “Concept Learning and Feature Selection Based on Square-Error Clustering”, Machine Learning 35, pp. 25-39, 1999.

[Neapolitan90] R. E. Neapolitan, “Probabilistic reasoning in Expert Systems, Theory and Algorithms”, Wiley, 1990.

[Ogden96] A. Ogden-Swift, “White Paper on Data Mining”, Honeywell Hi-Spec Solutions, UK, 1996.

[Pearl88] J. Pearl, “Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference”, Morgan Kaufmann, 1988.

[Pearl91] J. Pearl, T.S. Verma, “A theory of inferred causation”, Principles of Knowledge Representation and Reasoning: Proceedings of the 2nd International Conference (KR '91)', pp. 441–452, 1991.

[Pearl92] J. Pearl and T. Verma, “A Statistical Semantics for Causation”, Statistics and Computing 2, pp. 91-95, 1992.

[Pearson02] K. Pearson, A. Lee, “”, Biometrika, Vol. 2, No. 357, 1902-3.

[Petti90] T.F. Petti, J. Klein, P.S. Dhurjati, “Diagnostic Model Processor: Using Deep Knowledge for Process Fault Diagnosis”, AI in Chemical Engineering Journal, 36, pp. 565-575, 1990.

[Rabiner89] L. R. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition”, Proceedings of the IEEE, Vol. 77, No. 2, pp. 257-286, 1989.

[Rabiner93] L. R. Rabiner, B.-H. Juang, “Fundamentals of Speech Recognition”, Prentice-Hall, 1993.

[Sahami96] M. Sahami, “Learning Limited Dependence Bayesian Classifiers”, Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, pp. 335-338, 1996.

[Samad97] T. Samad, K. Lakshminarayan, “Data Mining for the Process Industries”, Honeywell Technology Centre, Minneapolis, USA, 1997.

[Settimi99] R. Settimi, J.Q. Smith, A.S. Gargoum, “Approximate Learning in Complex Dynamic Bayesian Networks”, Proceedings of the 15th Annual Conference on Uncertainty in AI, pp. 585-593, 1999.

[Shahar97] Y. Shahar, “A Framework for Knowledge-Based Temporal Abstraction”, Artifical Intelligence 90, pp. 79-133, 1997.

[Shortcliffe76] E.H. Shortcliffe, “Computer-Based Medical Consultation: MYCIN”, Amsterdam: Elsevier Science, 1976.

[Shumway91] R.H. Shumway, D. S. Stoffer, “Dynamic Linear Models with Switching”, Journal of the American Statistical Association 86, pp. 763-769, 1991.

[Snedecor67] G. Snedecor and W. Cochran, “Statistical Methods”, Iowa State University Press, 6th edition, 1967.

[Stirling1730] J. Stirling, “Methodus Differentialis”, pp. 137, 1730.

[Suzuki96] J. Suzuki, “Learning Bayesian Belief Networks Based on the MDL Principle: An Efficient Algorithm Using the Branch and Bound Technique”, International Conference on Machine Learning, pp. 462-470, 1996.

[Swift99a] S. Swift and X. Liu, “Modelling and Forecasting of Glaucomatous Visual Fields using Genetic Algorithms”, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99), pp. 1731-1737, 1999.

[Swift99b] S. Swift, A. Tucker and X. Liu, “Evolutionary Computation to Search for Strongly Correlated Variables in High-Dimensional Time-Series”, Proceedings of Intelligent Data Analysis 99, LNCS 1642, Springer-Verlag, pp. 51-62, 1999.

[Syswerda89] G. Syswerda, “Uniform Crossover in Genetic Algorithms”, Proceedings of the 3rd International Conference on Genetic Algorithms", Morgan Kaufmann, pp. 10-19, 1989.

[Thiesson98] B. Thiesson, C. Meek, D. M. Chickering, D. Heckerman, “Learning Mixtures of DAG Models”, Technical Report, MSR-TR-97-30, Microsoft Research, 1998.

[Tucker99] A. Tucker, X. Liu, “Extending Evolutionary Programming Methods to the Learning of Dynamic Bayesian Networks”, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99), pp. 923-929, 1999.

[Tucker2000] A. Tucker, S. Swift, N. Martin, X. Liu, “Grouping Multivariate Time Series Variables: Applications to Chemical Process and Visual Field Data”, The 20th SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence, 2000.

[Tucker2001a] A. Tucker, S. Swift, X. Liu “Variable Grouping in Multivariate Time Series via Correlation”, To appear in IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, 2001.

[Tucker2001b] A. Tucker, X. Liu, A. Ogden-Swift, “Evolutionary Learning of Dynamic Probabilistic Models with Large Time Lags”, To appear in The International Journal of Intelligent Systems, 2001.

[Ueda95] N. Ueda, R. Nakano, “Deterministic Annealing Variant of the SEM algorithm”, Advances in Neural Information Processing Systems 7, pp. 545-552, 1995.

[Waibel89] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, K. J. Lang, “Phoneme Recognition using Time Delay Neural Networks”, IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP-37, pp. 328-339, 1989.

[Wan90] E.A. Wan, “Temporal Backpropagation for FIR Neural Networks”, IEEE International Joint Conference on Neural Networks 1, pp. 575-580, 1990.

[Wong99] M. Wong, W. Lam, S. Leung, “Using Evolutionary Programming and Minimum Description Length Principle for Data Mining of Bayesian Networks”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 21, No.2, pp. 174-178, 1999.

Appendix A - Glossary

General

N	The Number of Variables in the domain
n	The length of the MTS
M	The order of a model
t	Time Position in MTS
A	The MTS
$A(t)$	The MTS vector at time t
a_i	Variable i
$a_i(t)$	Variable i at time t

Static BNs

S_h	Candidate Bayesian Network Structure
D	A Dataset (either static cases or time series)
X , x_i	The set of nodes, node i
π_i	The parent set of node i
r_i	The number of states on node i
$p(x_i \mid \pi_i)$	The probability of the instantiated node i given that its parents are in a particular instantiation
w_i	The states of all other nodes in a BN except x_i
DL	The Description Length of the Candidate Network $DL_{S_h} + DL_D$
DL_{S_h}	The Description Length of the Candidate Network Structure, S_h
DL_D	The Description Length of encoding the data, D, given S_h (Measure of Entropy)
DL_{π_i}	The Description Length of a node i, and its parents set
$\log p(D \mid S_h)$	Log Marginal Likelihood of a network structure
$\log p(D \mid \pi_i)$	Log Marginal Likelihood of a node i, and its parent set
MaxBranch	The maximum number of parents a node is allowed

Dynamic BNs

Q	The set of nodes in a DBN which represent variables at a time lag > 0
$\|Q\|$	The number of nodes in Q
t	The time slice of a set of nodes (also the position in an MTS)
MaxT	The maximum possible time lag over a dependency
(a_i, a_j, lag)	A triple representing a link from a parent representing variable

	a_i to a child node a_j with a time lag of *lag*
OpState	The Hidden Controller Node used to model changing Dependencies
H , h_i	Set of hidden nodes, hidden node i

Genetic Algorithms

Generations	Number of Generations
Popsize	Size of the Population (the number of individual chromosomes)
CrossoverRate	The probability of 2 parents being crossed over
MutationRate	The probability of a chromosome being mutated

Evolutionary Programming (for Seeding)

$gene_i$	Gene *i*. The *i*th element of a chromosome representing either a parent node, a child node or a lag
σ_i	Self Adapting Parameter relating to the standard deviation of a Normal Distribution
List	The list of highly dependant (strong correlated or good BN metric score) triples
R	The length of *List*

Grouping MTS Variables

G	The set of groups
g_i	The ith group
m	The number of groups
k_i	The size of the ith group
s	The Search Space
e	The number of explicit dependencies
c	The number of calls to a correlation coefficient / BN metric

Mathematical Functions and Distributions

$N(\mu,\sigma)$	Normal Distribution with mean μ and standard deviation σ
$U(min,max)$	Uniform Distribution with bounds of *min* and *max*, inclusive

Dynamic Cross Correlation Function

DCCF	A 2×2 matrix of correlations between 2 variables, where rows determine the window position over time and columns determine time lag between the variables
$\rho_{a_i a_j}(l, t_s, t_f)$	The DCCF value for time lag, l, between two variables, a_i and a_j for a window of data, delimited by t_s and t_f
win_{len}	The length of the window of data
win_{jump}	The size of shift made by the window of data during the calculation of the DCCF

Appendix B - Proofs for Grouping Evaluation Metric

Proof 1. When there are no correlations, then $List = \phi$. Therefore *max(f(G))* is 0, because there will never be any cases where *L* is 1. This therefore requires that the size of any of the groups in *G* will be 1. This is by definition of the functions *L* and *h*.

Proof 2. If a correlation exists for each pairing of variables, then the maximum size for *List* will be $\frac{N(N-1)}{2}$, because of the duplicate restriction. It therefore follows that the value for $h(g_i)$ will be $\frac{k_i(k_i-1)}{2}$ using the same logic. Using equation 6, we have

$$\max(f(G)) = \max(\sum_{i=1}^{m} h(g_i))$$

therefore

$$\max(f(G)) = \max\left(\sum_{i=1}^{m} \frac{k_i(k_i-1)}{2}\right)$$

since

$$\max\left(\sum_{i=1}^{m} \frac{k_i(k_i-1)}{2}\right) = \frac{1}{2}\max\left(\left(\sum_{i=1}^{m} k_i^2\right) - N\right)$$

then *f(G)* will be a maximum when $\sum_{i=1}^{m} k_i^2$ is a maximum.

We shall assume that $1 \le k_1 \le k_2 \le \ldots k_m$. If we write $k_1' = k_1 + k_2$ then

$$(k_1 + k_2)^2 = k_1^2 + k_2^2 + 2k_1k_2$$
$$\therefore (k_1')^2 > k_1^2 + k_2^2$$

This process can be repeated until there is only one value k_1 remaining where $k_1=N$, and f attains its maximum value. Hence when *List* is at a maximum size (as above), the arrangement with the maximum fitness will be all variables in a single group.

Proof 3. If the data generating the correlations came from a mixed set of multivariate time series observations, then for a given grouping arrangement G and correlation set *List*

$$\max(f(G)) = \sum_{i=1}^{m} \max(h(g_i))$$

$$\max(h(g_i)) = \max\left(\sum_{\substack{\forall a,b \\ a \neq b \\ 1 \leq a < b \leq k_i}} L(g_{ia}, g_{ib}) \right)$$

This will be a maximum when all instances of the function L are 1. If *List* contains an additional spurious correlation or is missing a correlation, then this value will be reduced by 1, by definition of L and proof 2. Hence the maximum value of the fitness for a given G will be when *List* contains the all of the correlations that can exist for each grouping.

Appendix C - The Lilliefors' Test Results

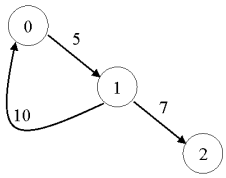
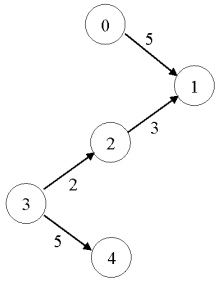
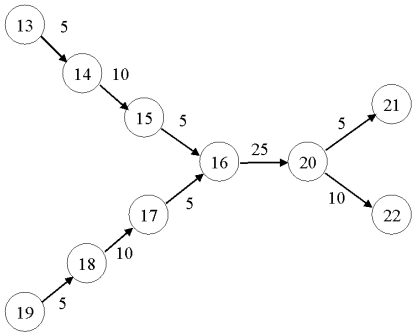
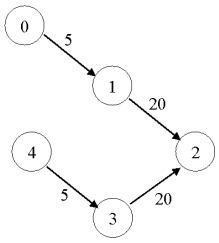
The table below displays a sample of the results of applying Lilliefors' test on a small sample of simulations as described in sections 4.1 and 4.2.

R	***c***	***S***	μ	σ	$\bar{\mu}$	$\bar{\sigma}$	D_{max}	$\frac{1.031}{\sqrt{v}}$	$D_{max} < \frac{1.031}{\sqrt{v}}$
50	657	4600	6.590	2.393	6.665	2.365	0.106	0.144	True
60	780	3900	10.823	2.939	10.909	3.152	0.079	0.132	True
70	357	3570	6.665	2.407	6.667	2.211	0.097	0.122	True
80	1062	7440	10.687	3.052	10.659	2.840	0.086	0.115	True
90	918	4590	16.293	3.639	16.364	3.629	0.069	0.108	True
100	1583	9500	15.400	3.589	15.382	3.420	0.070	0.103	True
110	1452	14520	10.418	3.058	10.476	2.846	0.082	0.098	True
120	1851	12960	15.972	3.743	15.997	3.476	0.067	0.094	True
130	1841	11050	19.95	4.138	19.993	3.896	0.060	0.090	True
140	2963	17780	21.539	4.222	21.536	4.055	0.059	0.087	True
150	3930	19650	27.127	4.767	27.273	4.581	0.052	0.084	True
160	2272	22720	15.222	3.708	15.238	3.640	0.071	0.081	True
170	4386	21930	30.896	5.007	30.909	4.898	0.053	0.079	True
180	1854	18540	17.158	3.878	17.143	3.957	0.065	0.077	True
190	988	9880	18.063	4.106	18.095	4.116	0.065	0.075	True

Within this table, μ, σ, $\bar{\mu}$ and $\bar{\sigma}$ are listed as examples for the section on Genetic Programming.

Appendix D - MTS Dataset Generation

Below is a selection of DBN structures that were used to generate the DBN datasets. Numbers associated with nodes represent variables and numbers associated with links represent time lags.

<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>DBN 1</td><td>DBN 2</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>DBN 3</td><td>DBN 4</td></tr>
</table>

Below is a selection of VAR process parameters that were used to generate the VAR datasets.

7 Variable VAR(3)																					
	P=1							P=2							P=3						
	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7
1	0.13	-0.32	-0.21	0.02	-0.22	-0.06	0.00	-0.29	0.11	0.30	-0.09	0.02	-0.04	-0.33	-0.05	0.17	-0.03	0.22	0.04	0.02	-0.25
2	-0.40	-0.63	0.02	-0.10	0.01	0.02	-0.22	0.05	0.16	0.58	0.05	-0.02	0.10	-0.24	-0.02	0.00	0.08	0.22	0.00	0.16	-0.01
3	0.11	-0.08	-0.10	0.03	-0.09	0.06	0.12	-0.12	-0.21	0.29	-0.17	0.08	-0.17	-0.26	0.09	-0.28	0.02	0.02	-0.03	-0.69	0.09
4	-0.11	-0.18	-0.42	-0.22	0.02	-0.13	0.12	-0.05	0.38	0.06	-0.14	-0.07	0.08	0.11	0.02	-0.27	-0.01	-0.08	0.32	0.02	0.06
5	-0.07	-0.01	0.09	0.31	0.08	-0.07	0.00	0.07	-0.61	-0.02	0.04	-0.29	0.01	-0.16	0.07	0.30	-0.41	-0.13	0.06	-0.15	0.06
6	0.06	0.27	-0.07	-0.19	0.16	-0.20	0.01	0.21	0.02	0.04	0.19	0.18	0.44	-0.04	0.06	0.18	-0.34	0.19	-0.05	-0.04	0.06
7	0.08	0.21	-0.32	0.07	-0.11	0.03	-0.41	-0.03	-0.33	-0.12	0.09	0.00	0.55	0.19	0.14	-0.02	0.02	0.16	0.04	-0.15	0.04

3 Variable VAR(5)															
	P=1			P=2			P=3			P=4			P=5		
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
1	0.10	0.02	-0.20	0.15	-0.04	0.24	0.06	-0.08	0.31	-0.07	0.03	-0.04	-0.05	0.19	0.15
2	-0.16	-0.02	-0.02	0.01	-0.40	0.10	-0.28	0.02	-0.24	-0.08	-0.01	-0.04	0.12	0.06	0.19
3	0.03	0.55	-0.18	0.08	-0.02	0.04	-0.09	0.00	0.19	0.05	0.26	0.01	-0.12	-0.02	-0.23

2 Variable VAR(2)				
	P=1		P=2	
	1	2	1	2
1	-0.52	0.02	0.23	-0.22
2	0.28	-0.17	-0.41	-0.11

Appendix E - Genetic Programming Details

Operator	Description	Value
Population	Constant Population	100
Generations	Number of iterations	μ =1,000 σ =50,000
Crossover	Percentage of population allowed to breed	0.75
Prune Mutation Rate	Cut down a sub-tree to a random terminal node	0.25
Add Sub-tree Mutation Rate	Replace a sub-tree for a new random sub-tree (size varies)	0.25
Change Node Mutation Rate	Change an operator to a new random operator or a terminal symbol to a new random terminal symbol	0.25
Survival	Simply select the top "Population" after new individuals have been added through Crossover and Mutation	Deterministic and Extinctive

μ and σ Results

The test dataset of parameters for the problems consisted of the same 150 records generated by the simulation experiments. This was divided into two halves, one for the training set and the other for the verification set. The records were numbered from 1 to 150, and μ was trained on the even records and σ on the odd records. The magnitude of the values for *R, s* and *c* increased as the record identifier (ID) increased.

Results were as follows:

Result	μ		σ	
Value for function (1)	$\frac{2cR^2}{2Rs+c(R+4)}$		$\frac{R+8}{63}+\frac{11c}{s}$	
Approximation to be used (2)	$\frac{2cR}{2s+c}$		$\frac{R}{63}+\frac{11c}{s}$	
	(1)	(2)	(1)	(2)
Fitness for Training Set (Excluding Nodes)	-0.1037	-0.2403	-0.7906	-2.0954
Sum of Absolute Error for Training Set	2.2198	3.1939	6.0883	11.3506
Average % Error for Training Set	0.2029	0.2512	2.3911	4.4791
Fitness for Testing Set (Excluding Nodes)	-1.038	-0.2490	-0.8421	-2.1139
Sum of Absolute Error for Testing Set	2.2355	3.1997	6.2628	11.4010
Average % Error for Testing Set	0.8148	1.1119	2.4604	4.5181

Appendix F - Dynamic Cross Correlation Functions

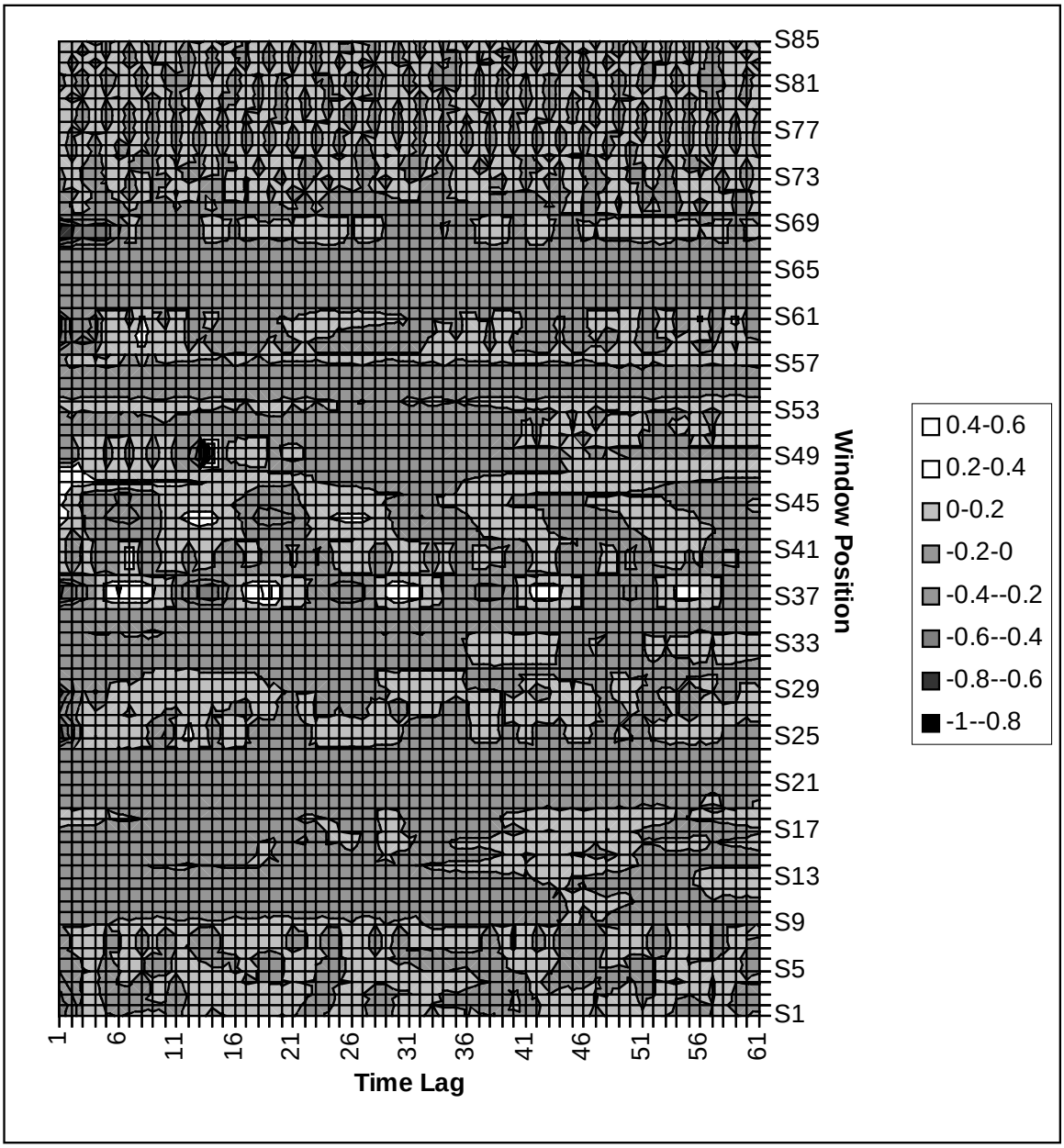


Figure F.1(a). The DCCF corresponding to AUTO/Man Girb with TGF, (win_{len} = 1000, win_{jump} = 500)

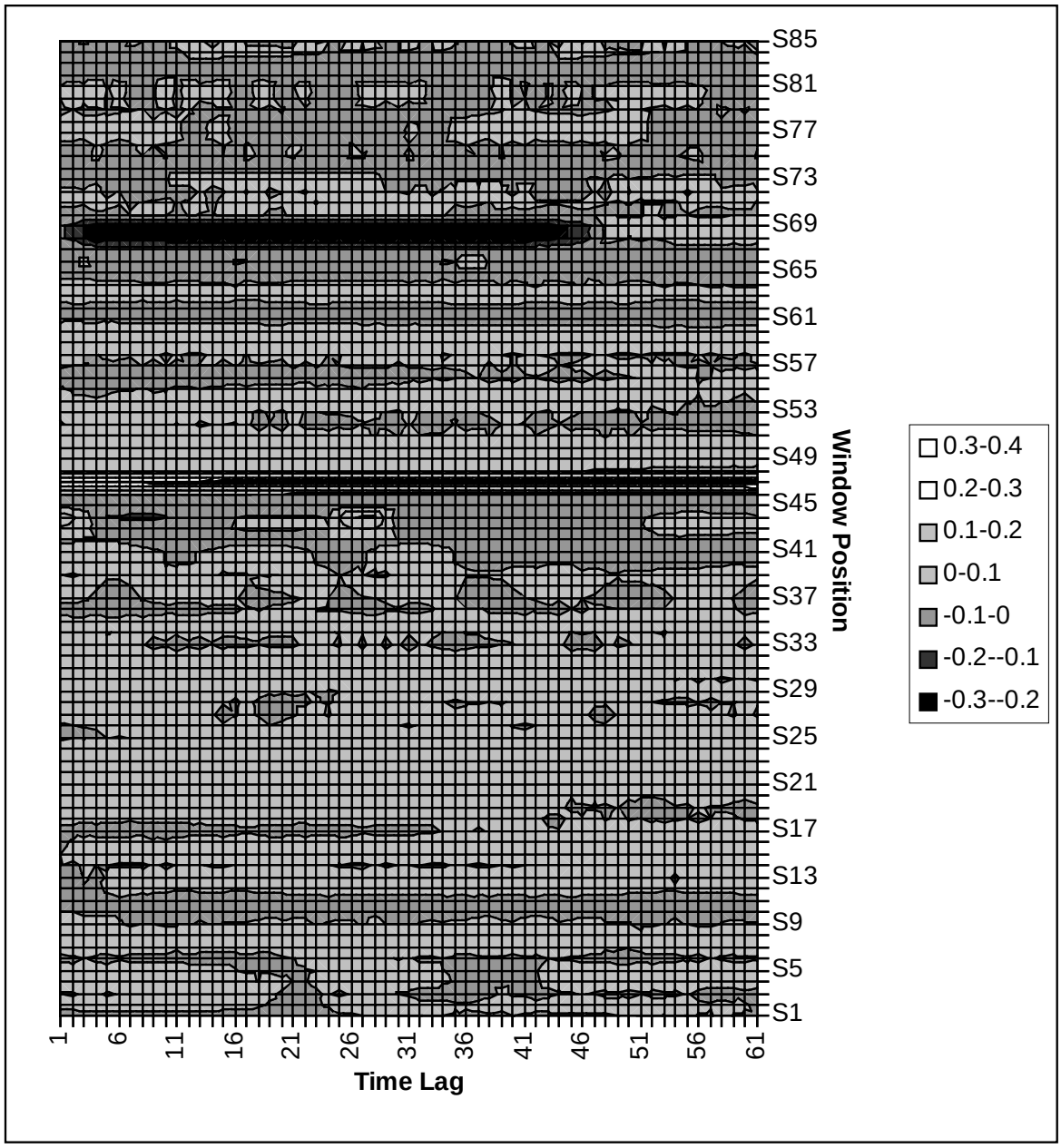


Figure F.1(b). The DCCF corresponding to SOT with TGF
(win_{len} = 1000, win_{jump} = 500)

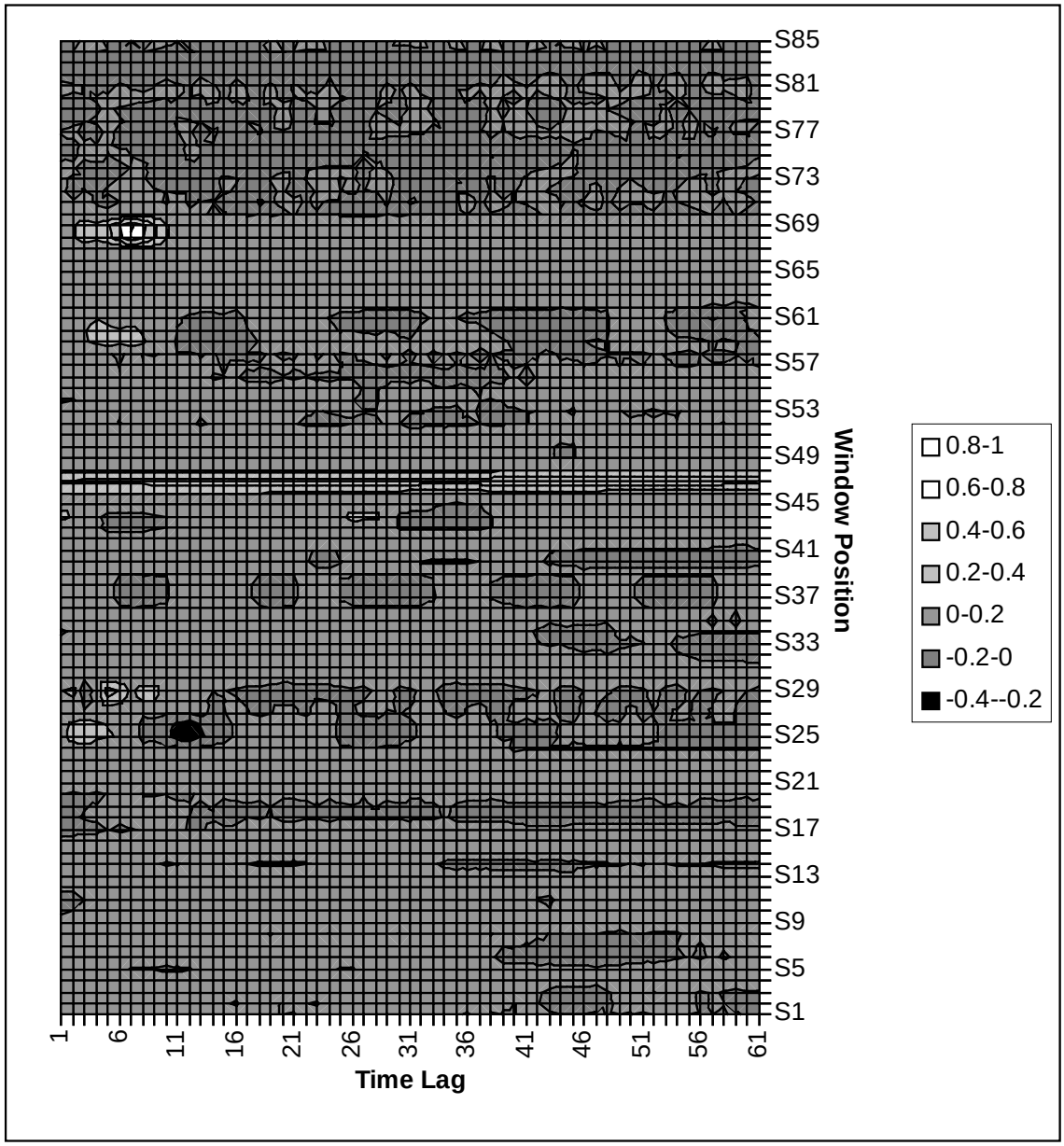


Figure F.1(c). The DCCF corresponding to T6T with TGF.
(win_{len} = 1000, win_{jump} = 500)

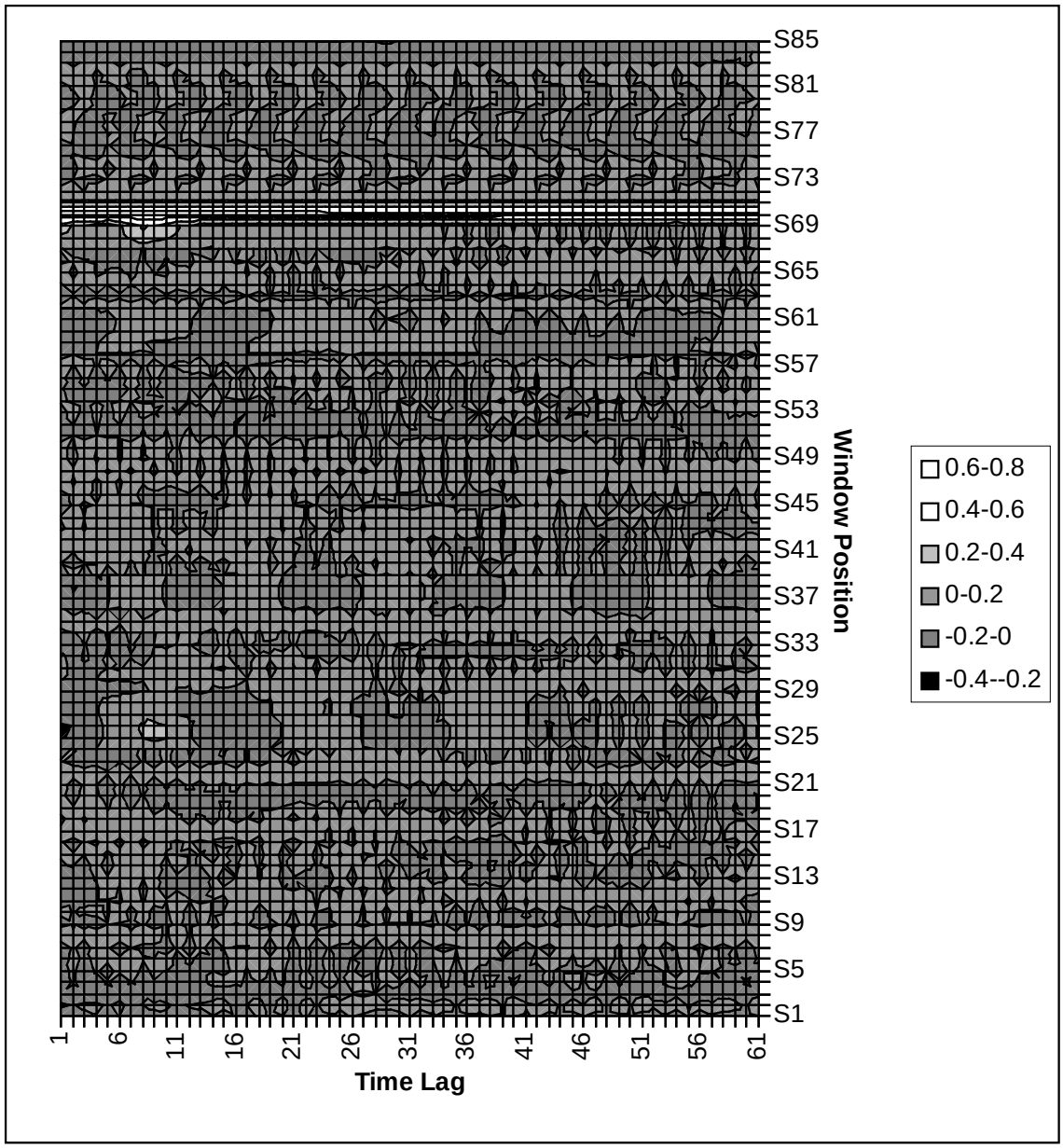


Figure F.2(a). The DCCF corresponding to RBT with BPF,
(win_{len} = 1000, win_{jump} = 500)

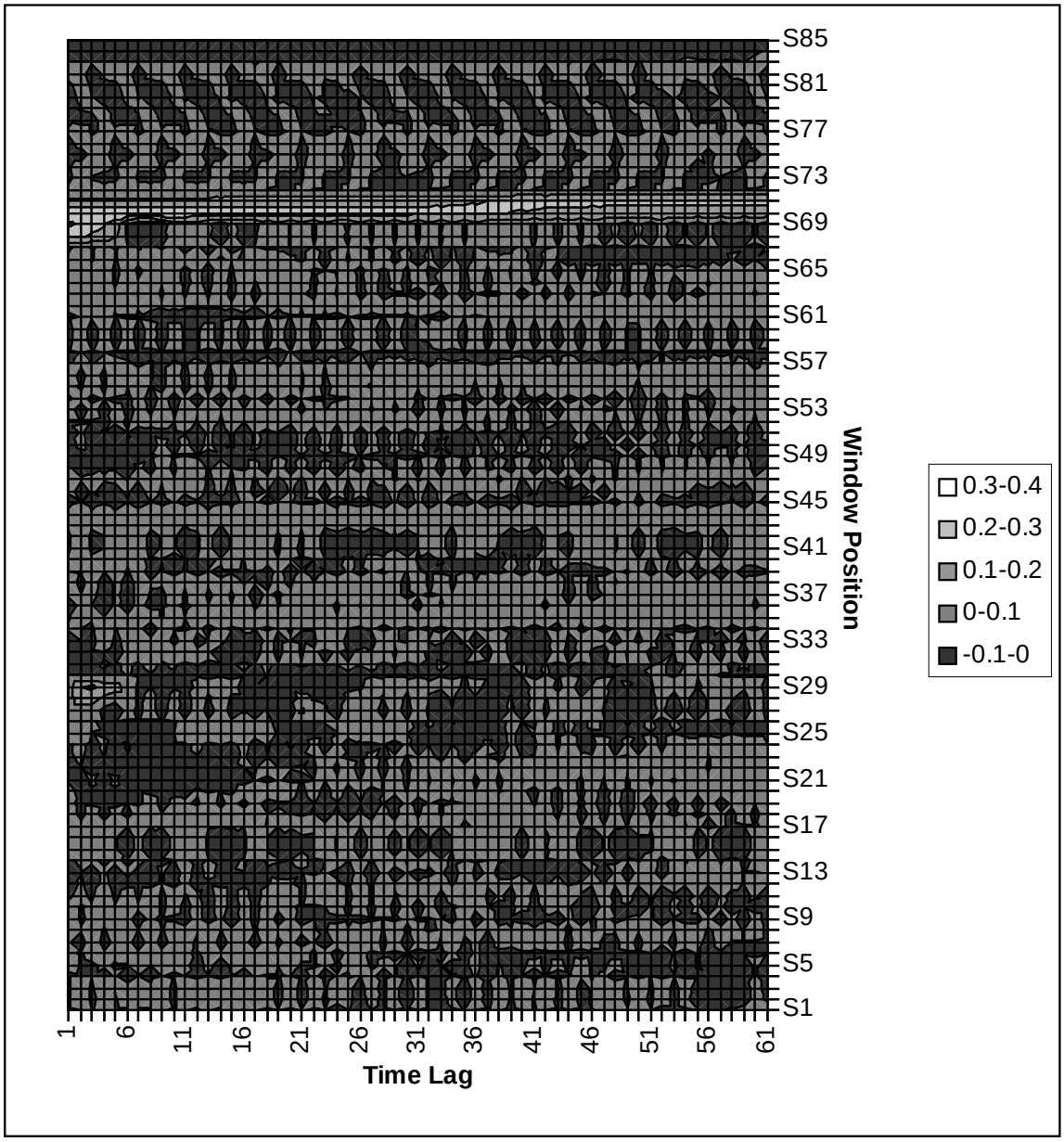


Figure F.2(b). The DCCF corresponding to C11/3 Inlet with BPF
(win_{len} = 1000, win_{jump} = 500)

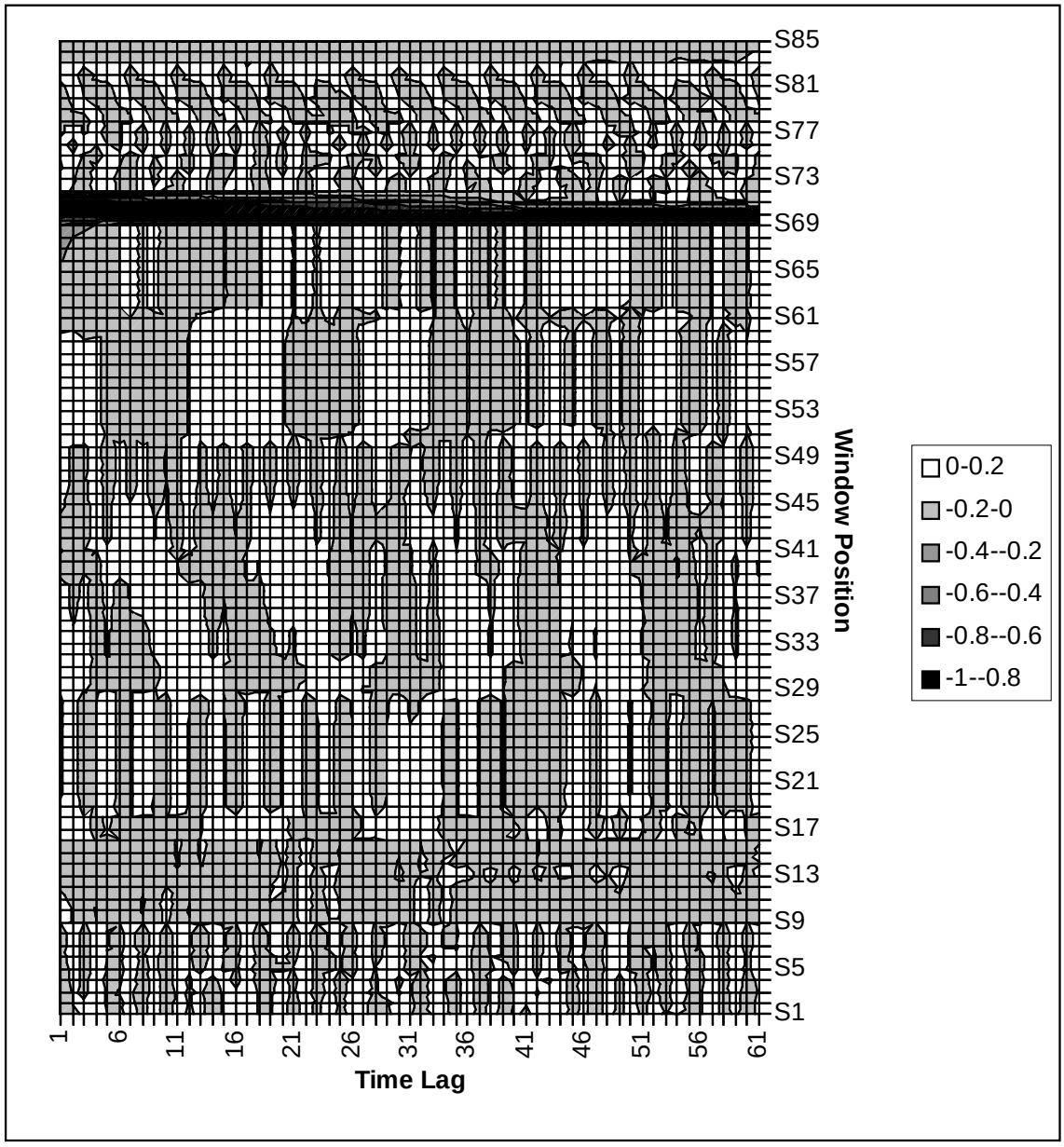


Figure F.2(c). The DCCF corresponding to AUTO/MAN Gas with BPF (win_{len} = 1000, win_{jump} = 500)

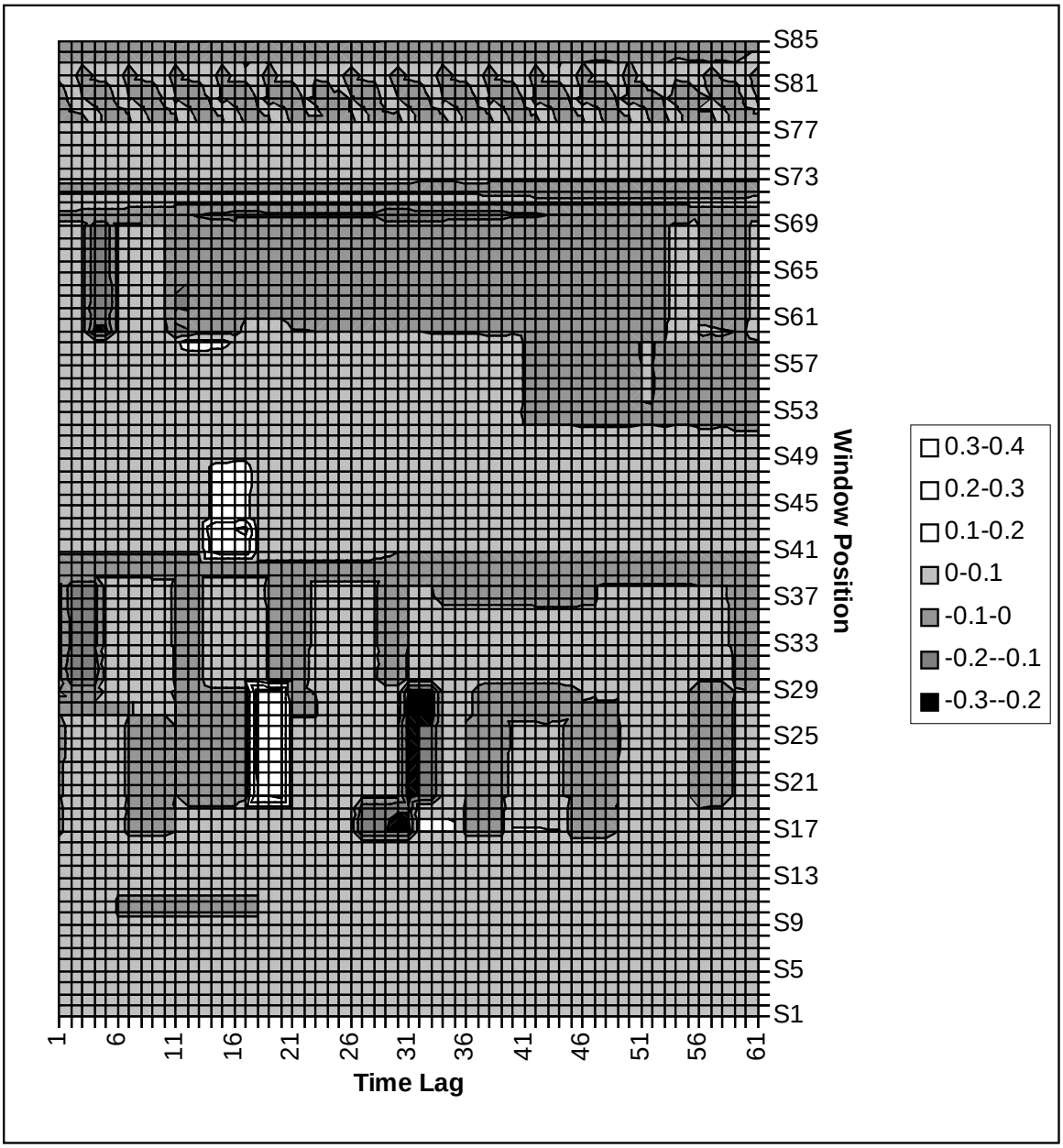


Figure F.3(a). The DCCF corresponding to %C3 with T36T
(win_{len} = 1000, win_{jump} = 500)

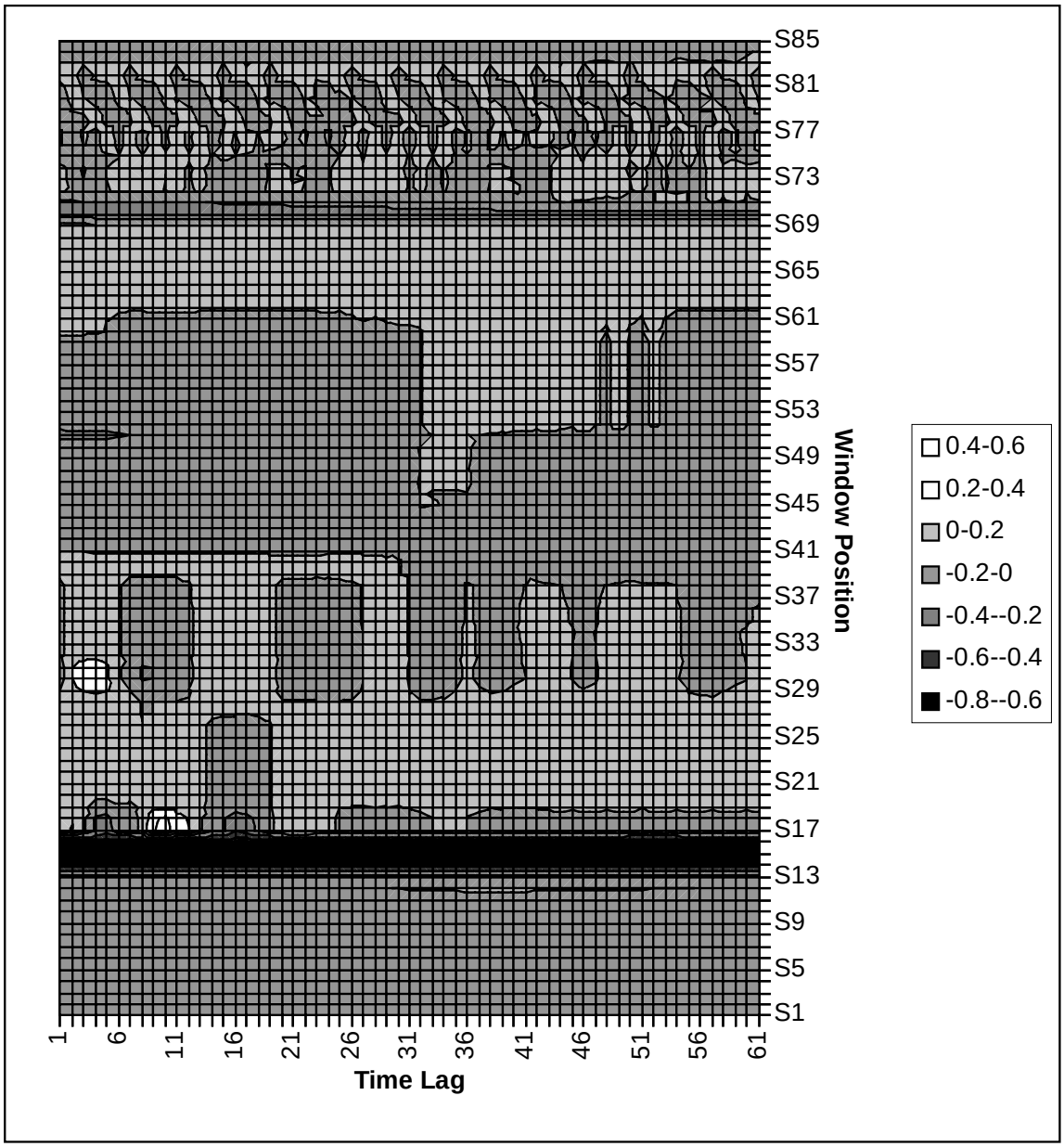


Figure F.3(b). The DCCF corresponding to A/M_GB with T36T
(win_{len} = 1000, win_{jump} = 500)

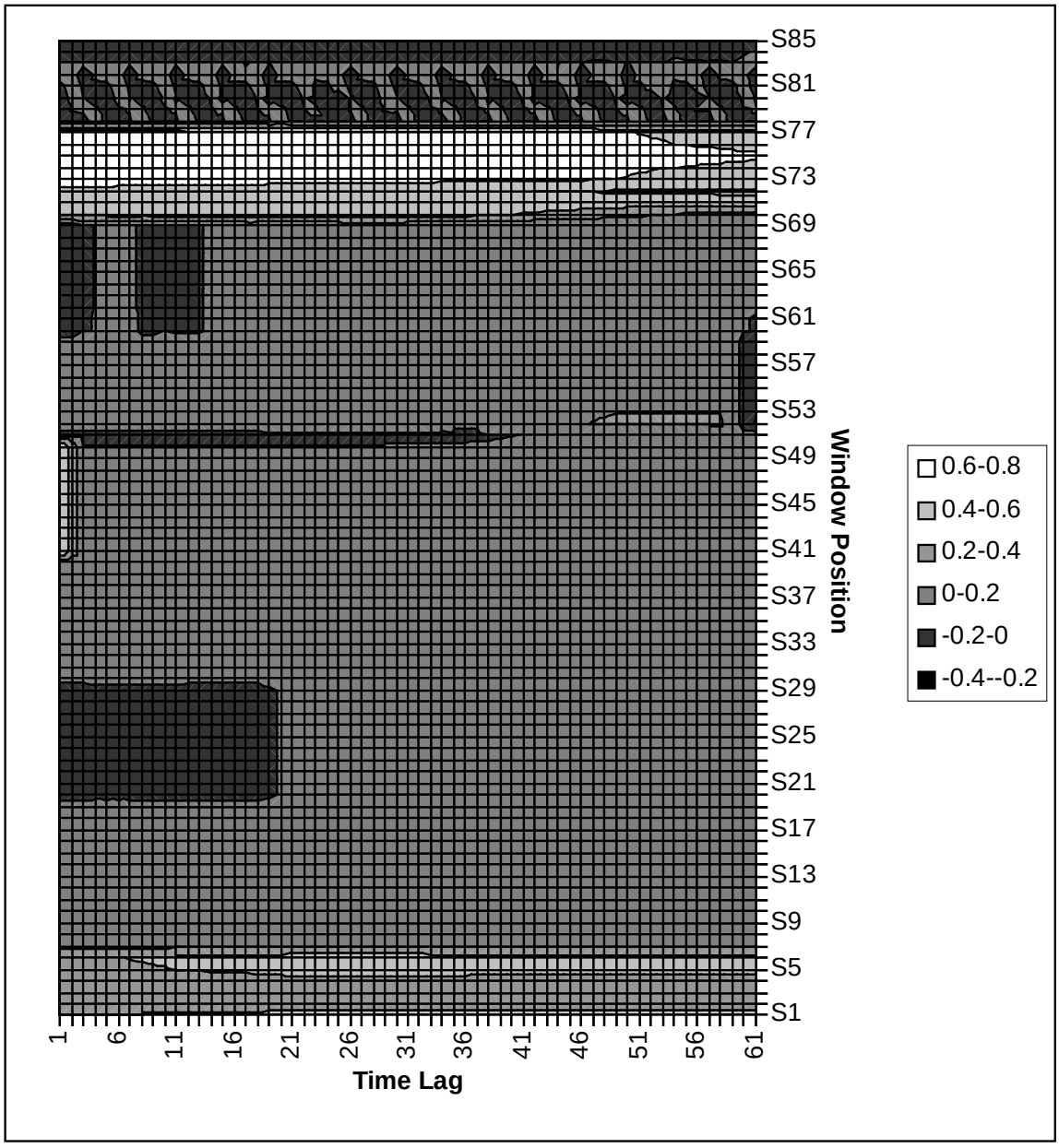


Figure F.3(c). The DCCF corresponding to AFT with T36T
(win_{len} = 1000, win_{jump} = 500)

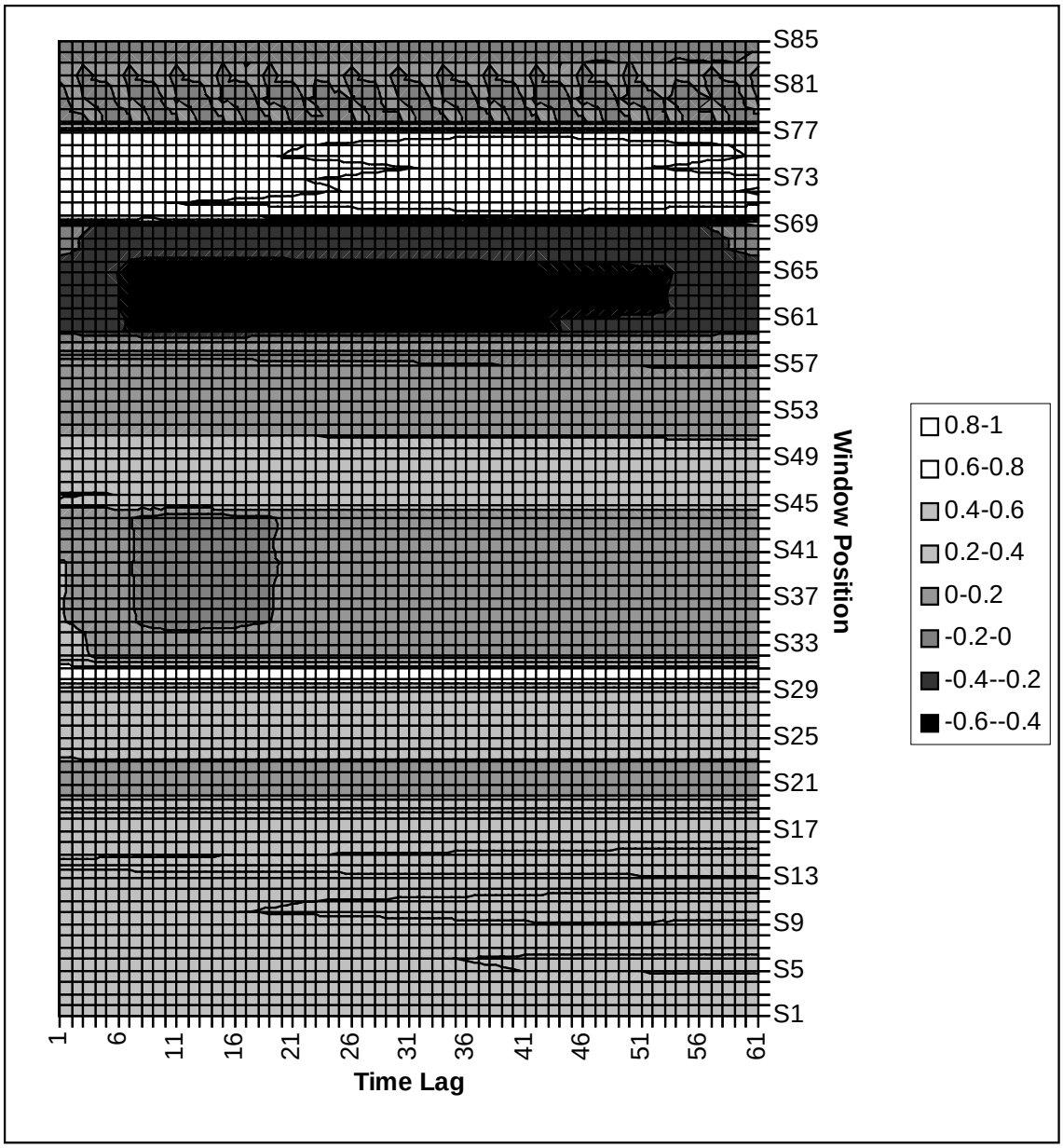


Figure F.4(a). The DCCF corresponding to SOT with TT
(win_{len} = 1000, win_{jump} = 500)

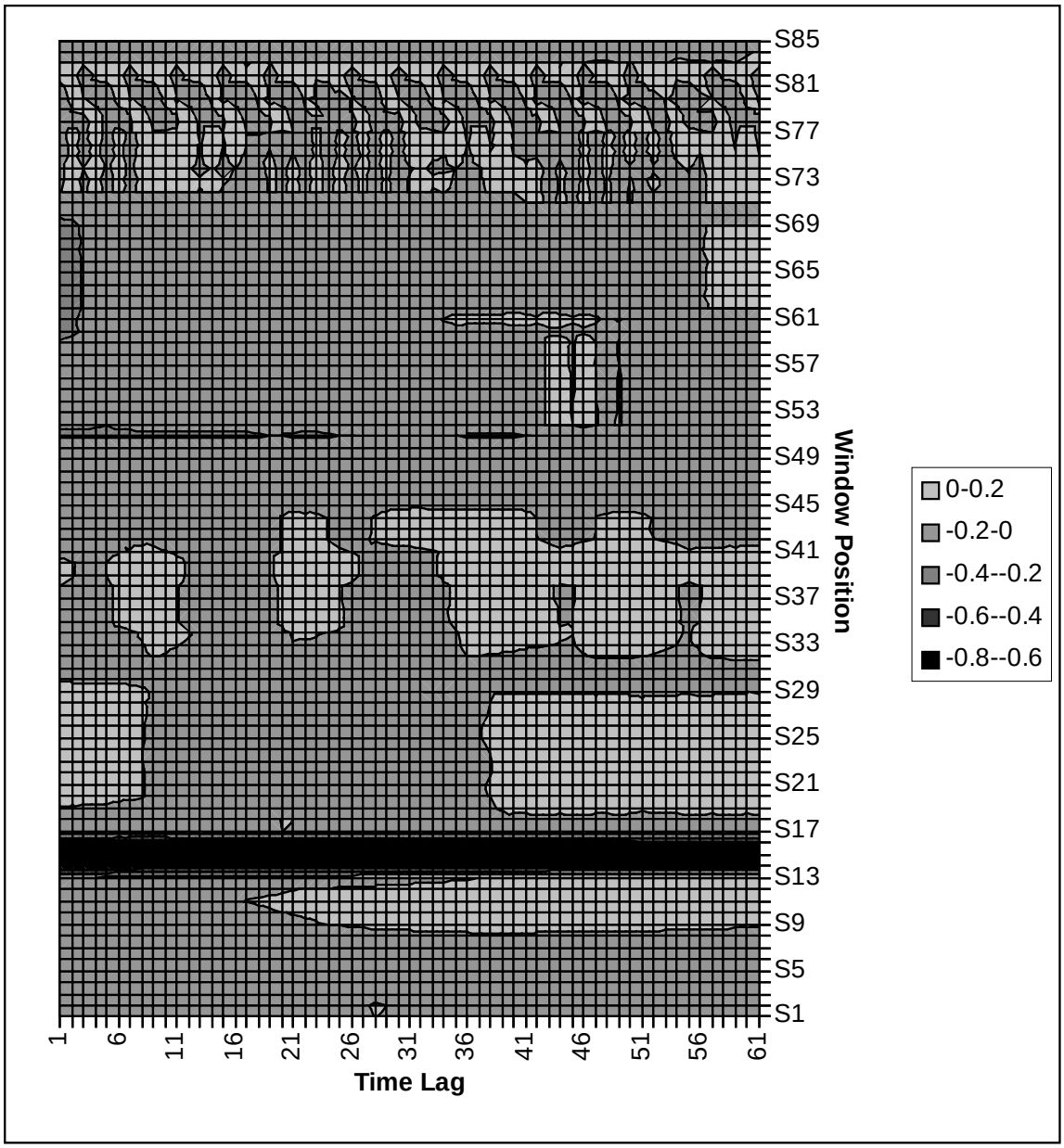


Figure F.4(b). The DCCF corresponding to A/M_GS with TT
(win_{len} = 1000, win_{jump} = 500)

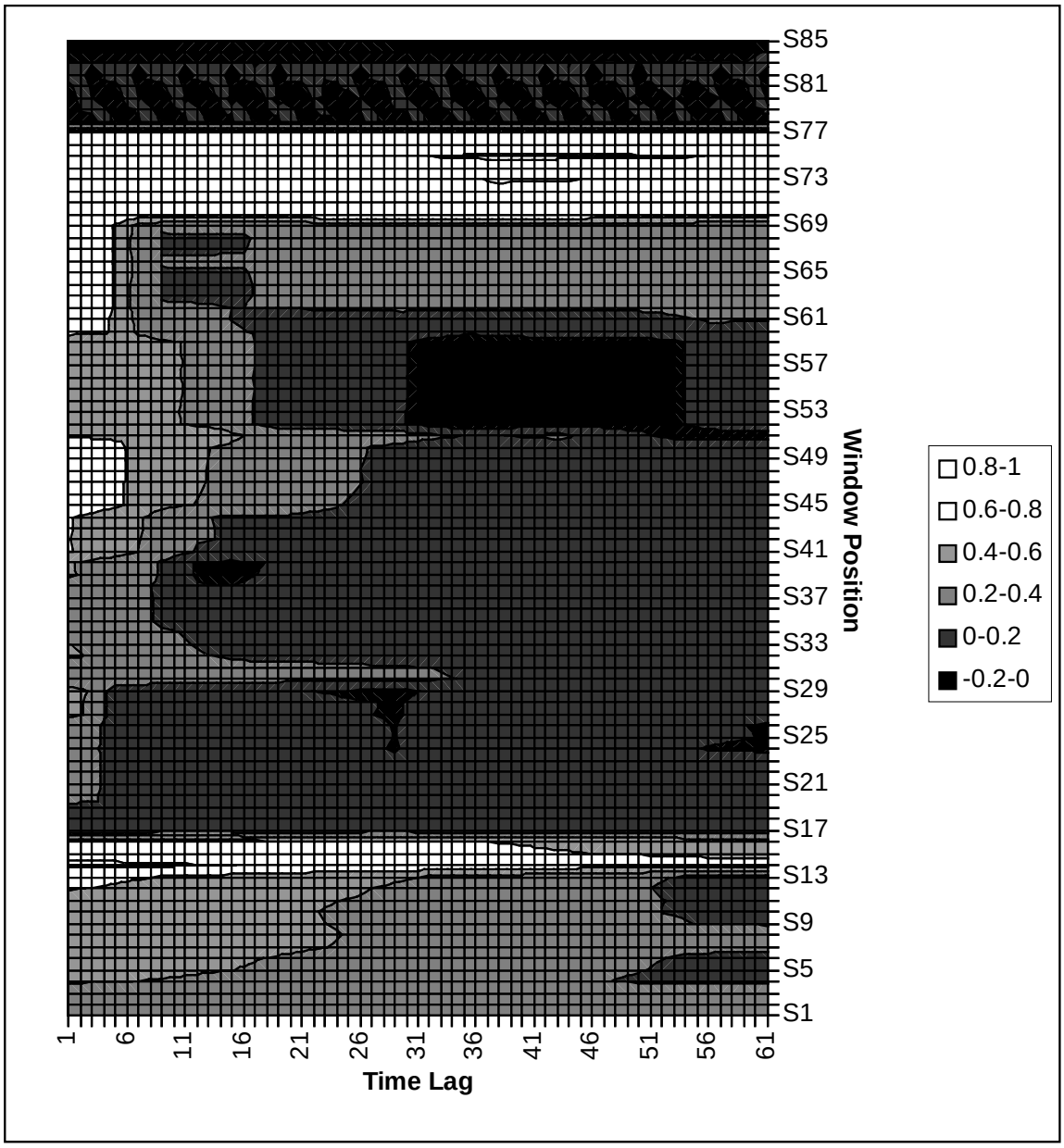


Figure F.4(c). The DCCF corresponding to T6T with TT
(win_{len} = 1000, win_{jump} = 500)

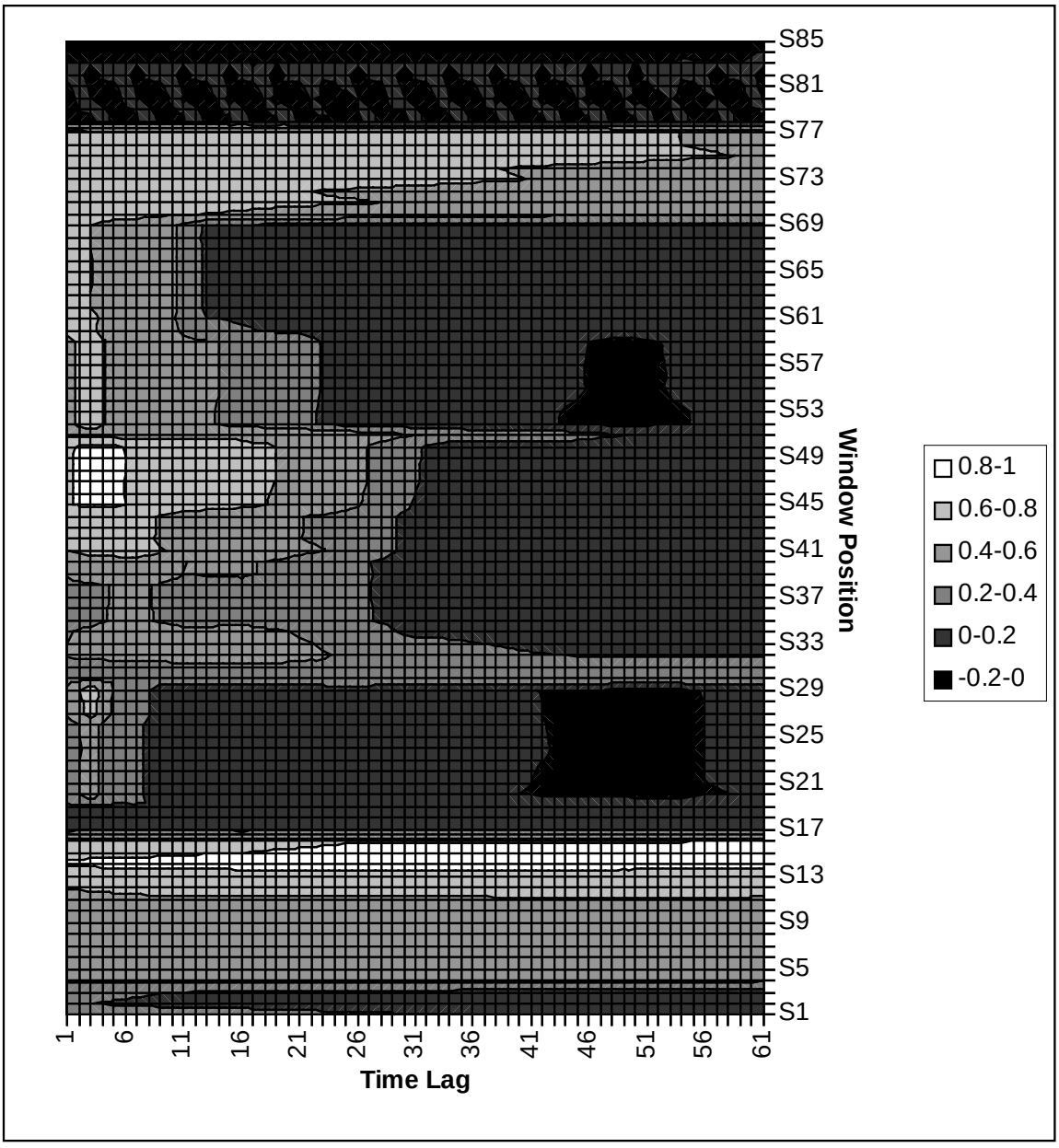


Figure F.5(a). The DCCFs corresponding to TT with T6T
(win_{len} = 1000, win_{jump} = 500)

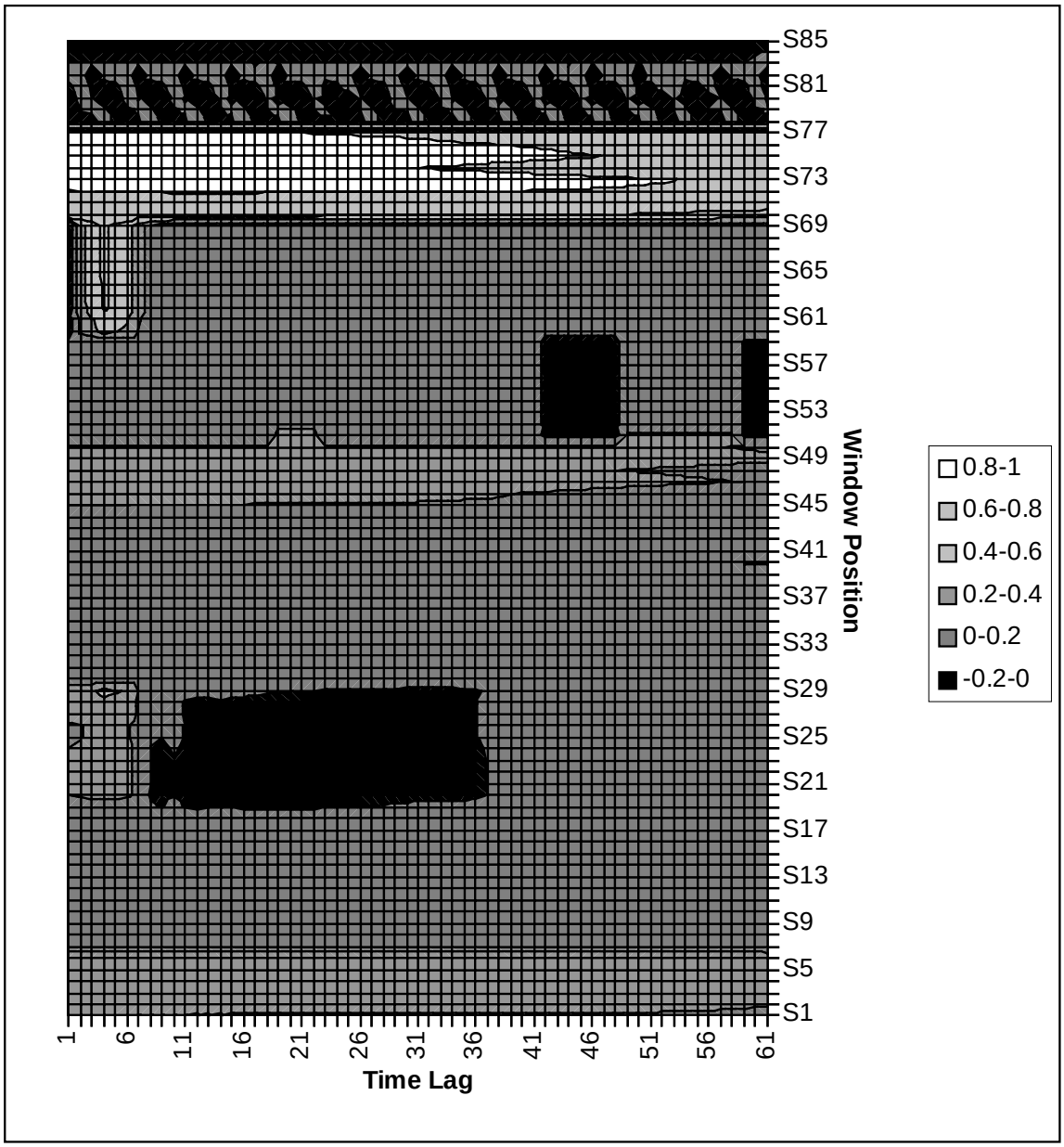


Figure F.5(b). The DCCF corresponding to C11/3 with T6T
(win_{len} = 1000, win_{jump} = 500)

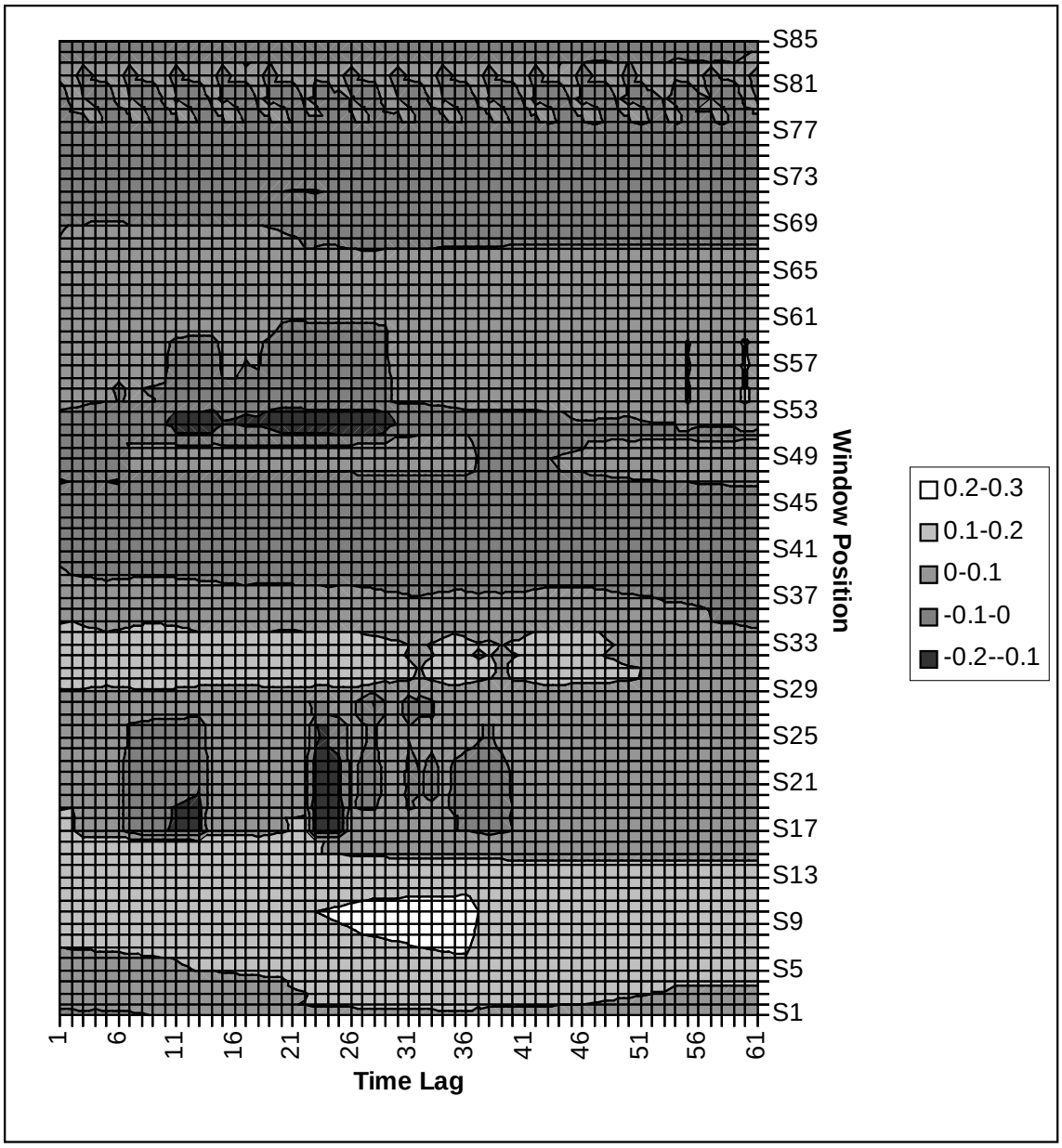


Figure F.5(c). The DCCFs corresponding to %C2 with T6T
(win_{len} = 1000, win_{jump} = 500)

Appendix G - Structural Expectation Maximisation for the Explanation of Multivariate Time Series with Changing Dependencies

The Structural Expectation Maximisation (SEM) algorithm which was introduced in Chapter 2 provides a method for learning hidden variables using a two-stage, iterative algorithm. The SEM algorithm [Friedman97/98b] which tackles the problem of learning structure from data with missing values or hidden variables, can be adapted to solve the problem of modelling changing dependencies. In these experiments, a more general case of this algorithm was investigated where the expected likelihood is not maximised at each iteration but merely incremented.

The SEM Algorithm

The algorithm uses the architecture in Figure G.1 and involves an iterative process whereby the network (current model) is initialised with a random structure and a random set of parameters for the hidden variables (here the *Op_States*). The current model and training data are then used to generate the expected statistics for the hidden variables. In the context of Bayesian network parameters, this involves applying inference using the current model where the training data supplies the observations. The resulting posterior distributions over the hidden nodes are then used to determine their expected states given the training data. Given the expected statistics and the current model, a search engine is used to improve the log likelihood score of the network structure. This process of repeatedly calculating expected

statistics and structure search continues until some stopping criteria is met such as minimal improvement in score or the maximum number of iterations is met.

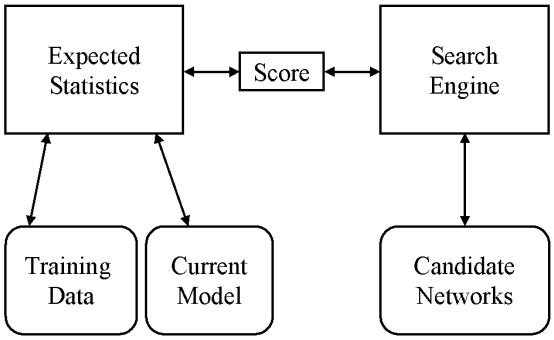


Figure G.1. The SEM Architecture (from [Friedman97]).

Algorithm G.1 which has been adapted from Algorithm 2.4 in Chapter 2 to determine the structure and values of *OpState*. It also uses a *DBN_List* in the same way as the HCHC from Chapter 6.

```
I/P  Random initial structure, S_h, random parameters except
     from observed variables, A, Iterations, MaxT
1    i = 0
2    Repeat until convergence or i > Iterations
         Expectation Step
3        Use inference to calculate the expected statistics
         for OpState given the current structure and values
         for OpState.
4        Use these statistics to reassign the values for
         OpState based upon the observed values in the MTS
         Maximum Likelihood Step
5        Search for structure that improves the expected
         score given the new values of OpState using a
         standard scoring metric such as log likelihood or DL
6        i = i + 1
7    End Repeat
O/P  Final Structure and Parameters
```

Algorithm G.1 - The SEM Algorithm for Learning DBNs with Changing Dependencies.

Applying SEM to Synthetic Data

The SEM algorithm has been applied to the synthetic datasets in Chapter 6. The average log likelihood at convergence is shown in Table G.1 along with the results of applying HCHC from Chapter 6. It is apparent that the log likelihood of the DBNs constructed with the SEM method are much higher than those resulting from HCHC. This could be due to HCHC being limited in its number of segmentations whereas the SEM is not. However, as the true number of segmentations for each dataset was within the limits set for HCHC, this implies either an inefficient HCHC or overfitting in the SEM. We investigate these hypotheses in the remainder of this Appendix.

	HCHC	SEM
MTS1	-3115.665	-220.738
MTS2	-15302.017	-2032.371
MTS3	-4494.706	-239.346

Table G.1. Resulting Log Likelihoods Upon the Synthetic Data.

i) Structure

The results in Table 6.3 display the Structural Difference (SD), as used in Chapter 5 and Chapter 6, between the original and the discovered structures. They are somewhat disappointing. It appears that many links were missed. However, a number of implicit dependencies were discovered which may help to explain the relatively good scores in Table G.1. The reason for the high number of missed links could be for several reasons such as bad segmentation which is looked at next. Also, there are a large number of very strongly correlated variables in the synthetic data, resulting in many implicit dependencies. This is

likely to be true of many MTS data and so the effect of this on learning models for explanation is worth considering in the future.

	MTS1	MTS2	MTS3
Number of Original Links	12	26	16
Spurious	4.2	3.0	1.0
Implicit	2.1	5.5	1.6
Missed	3.3	8.3	8.2
Total SD	9.6	16.8	10.8

Table G.2. Structural Difference Results using SEM.

ii) Segmentation

The expected states of the hidden variables *OpState* were explored to see how they changed over time. Ideally this would reflect the segmentations of the synthetic datasets. The breakdown of MTS 3 is now examined in more detail. The original structures and partitions used to generate this dataset are as shown in Figure 6.5 of Chapter 6.

Figures G.2 - G.6 display the resulting *OpState* variables for each of the 5 segments within MTS 3. Whilst the nature of the *Op_State* variable's behaviour appear to change within each segment (for example in one segment they may remain steadily in one state whilst in another they may fluctuate rapidly between 2 or more), they do not come close to neatly allocating a different state to different MTS segments. The problem in distinguishing between different states on this synthetic data is likely to be for two reasons. Firstly, the data has been generated so that there are many tightly related variables and changes in dependency such that a link from a_0 to a_1 can switch to a link from a_1 to a_0. This will mean that for a lot of the data, the expected state will be the same for both these situations (because there is a strong correlation between two variables irrespective of direction due to spurious correlations). It is

not known quite how much this will be true within the real datasets although many tightly related variables are expected. Secondly, SEM is well known to suffer from local maxima and although perturbation has helped reduce this, a more drastic approach may be required such as deterministic annealing [Ueda95]. It may be the case that the better results attained from HCHC were due to the assumption that *OpStates* would be relatively stable, thus preventing overfitting to the data.

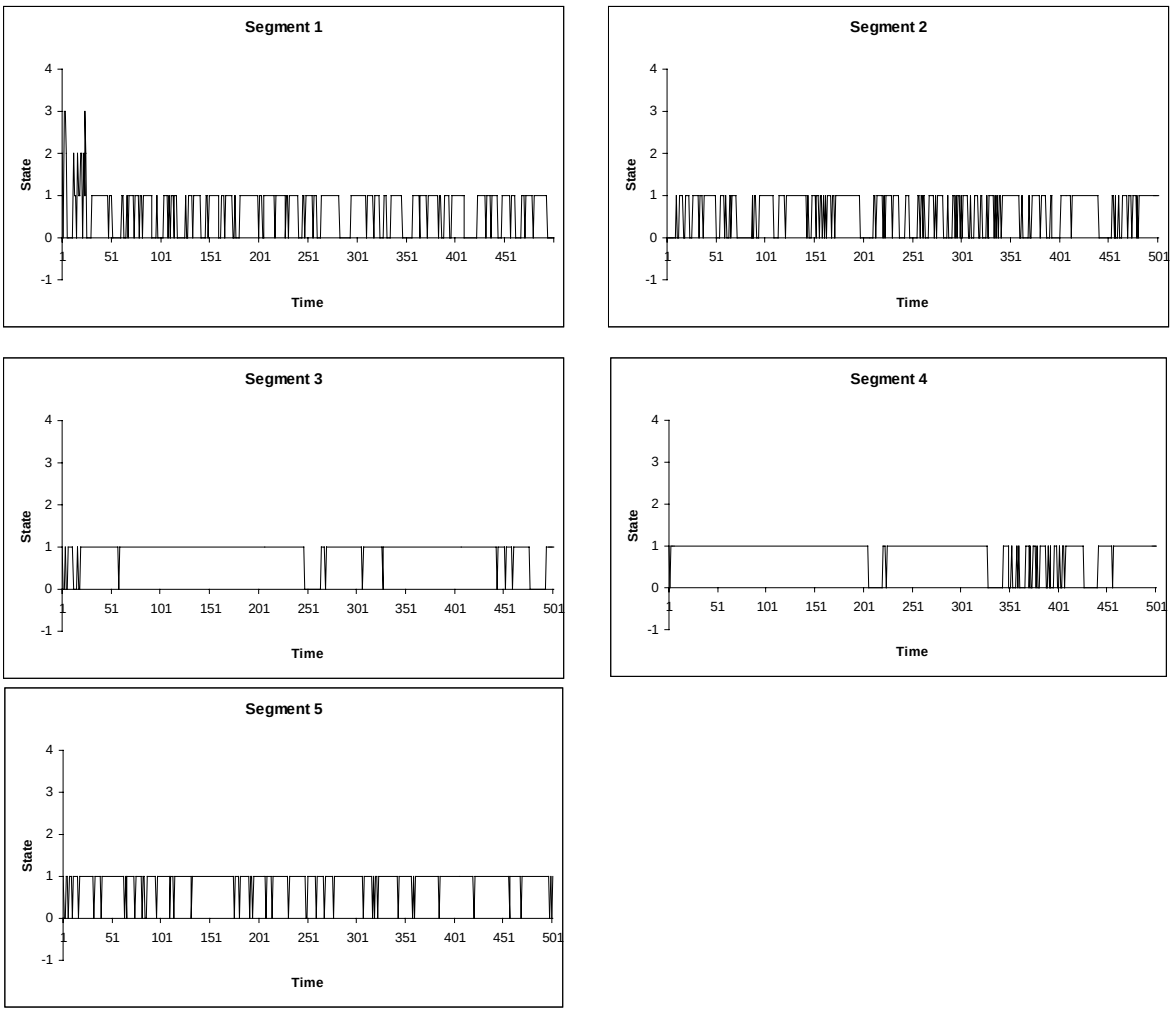


Figure G.2. Expected states for $OpState_0$ in each of the 5 segments in MTS 3

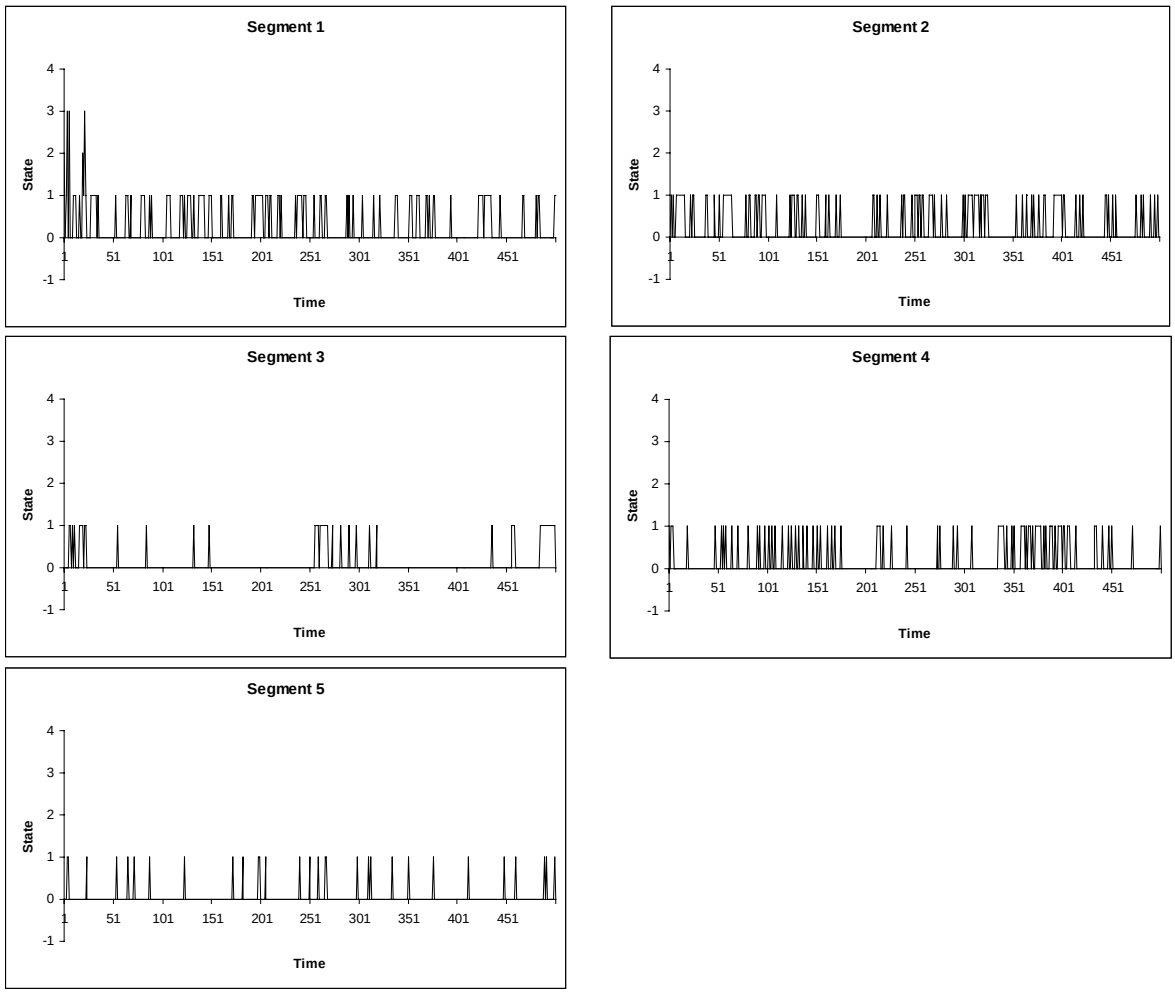


Figure G.3. Expected states for $OpState_1$ in each of the 5 segments in MTS 3

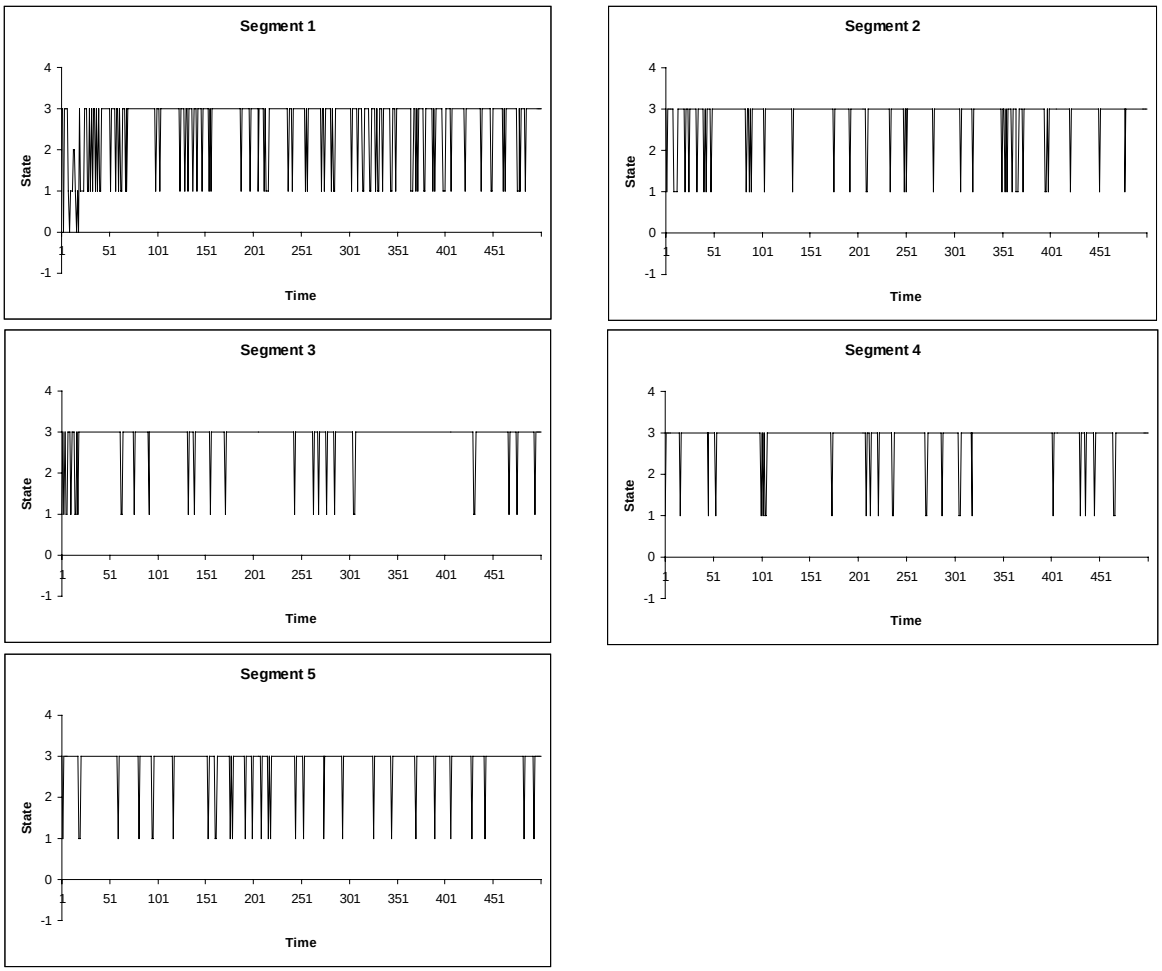


Figure G.4. Expected states for $OpState_2$ in each of the 5 segments in MTS 3

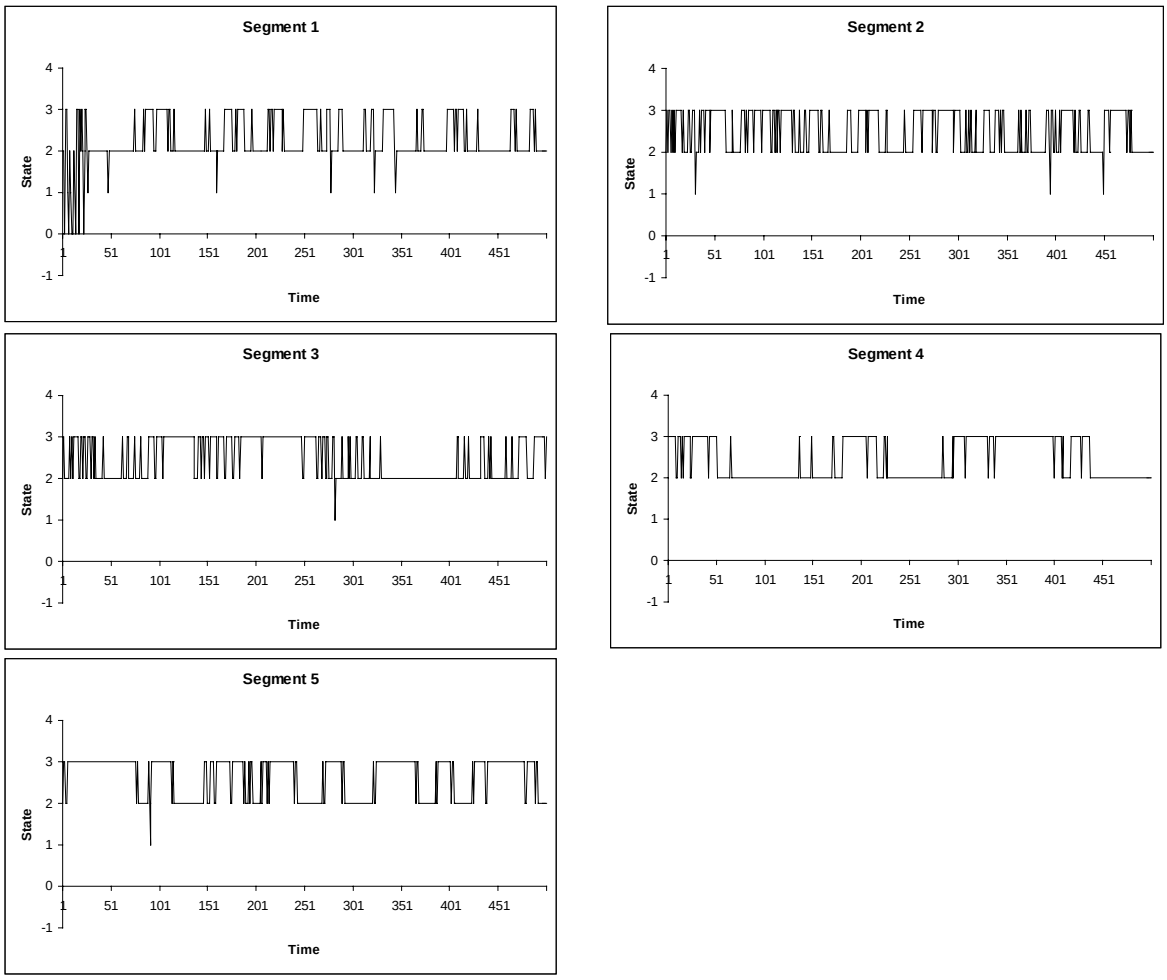


Figure G.5. Expected states for $OpState_3$ in each of the 5 segments in MTS 3

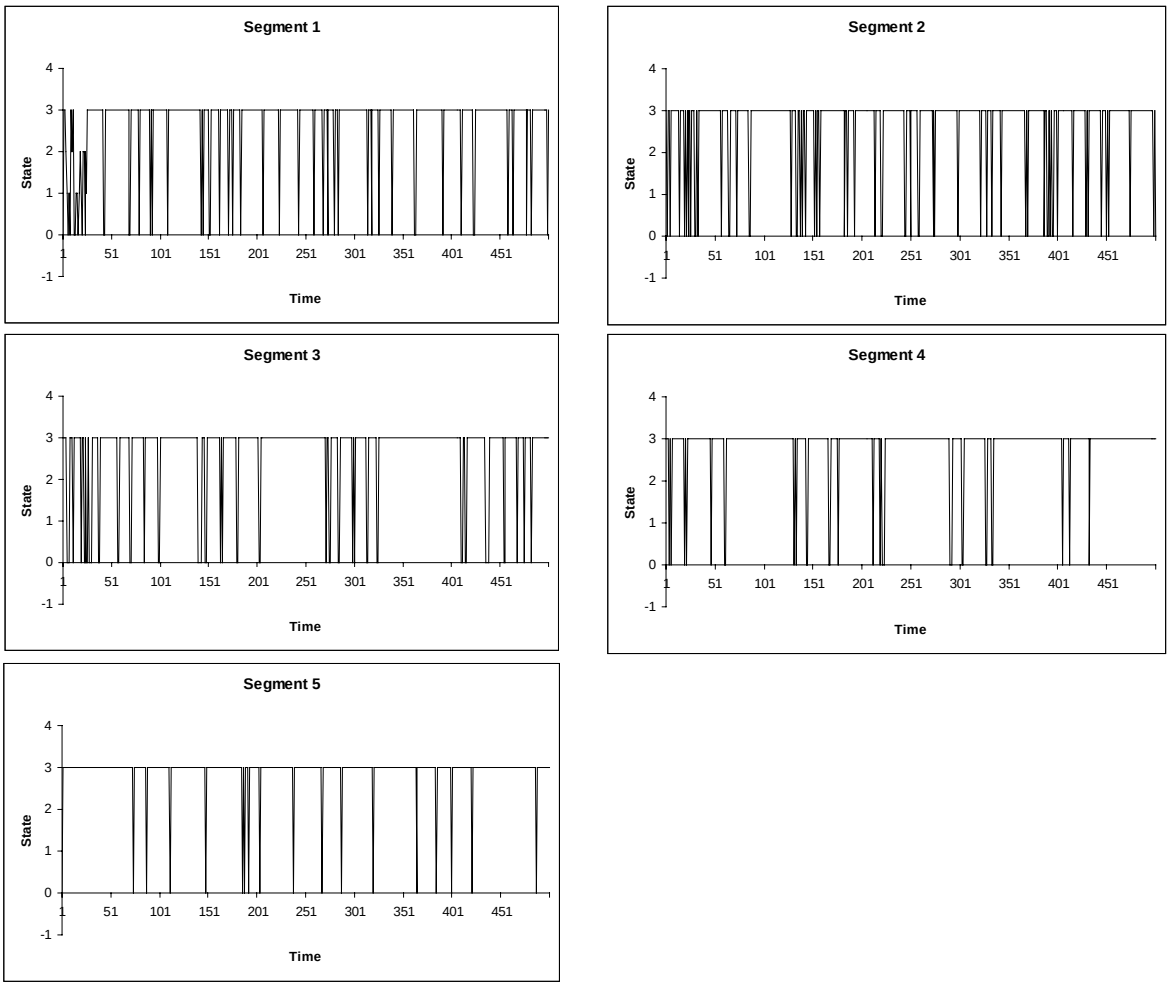


Figure G.6. Expected states for $OpState_4$ in each of the 5 segments in MTS 3