Evolutionary Learning of Dynamic Probabilistic Models with Large Time Lags

Allan Tucker[1], Xiaohui Liu[1], Andrew Ogden-Swift[2]

[1]Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex. UB8 3PH, UK. {Allan.Tucker, Xiaohui.Liu}@Brunel.ac.uk

[2]Chilworth Research Centre, Honeywell Hi-Spec Solutions, Southampton.S016 7NP, UK. Andrew.Ogden-Swift@uk.Honeywell.com

In this paper, we explore the automatic explanation of Multivariate Time Series (MTS) through learning Dynamic Bayesian Networks (DBNs). We have developed an evolutionary algorithm which exploits certain characteristics of process MTS in order to generate good networks as quickly as possible. We compare this algorithm to other standard learning algorithms that have traditionally been used for static Bayesian networks but are adapted for DBNs in this paper. These are tested on both synthetic and real-world MTS. We evaluate sample explanations which have been generated from chemical process data using our methodology, and several useful heuristics, we have found that the proposed method is more efficient for learning DBNs from MTS with large time lags, especially in time-demanding situations.

1. INTRODUCTION

Many complex dynamic processes record Multivariate Time-Series (MTS) data at frequent time periods. These data will be characterized by a large number of interdependent variables, though some may have no substantial impact on any others. There can be large time lags between causes and effects. Take the oil refinery process as an example. In reviewing oil refinery data, 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 they occur, which could then be used to modify operating practices, retrain operators or conduct anticipatory planning.

In order to explain events, some method is required for reasoning about relationships between these variables back in time. For example, the reason for a particular temperature becoming extremely high may be that a flow rate dropped ten minutes ago and the flow dropped because, one minute before that, a valve was closed by a control engineer. A well-known paradigm for performing probabilistic inference about a system is the Bayesian Network [15, 21] and its dynamic counterpart can model a system over time [7, 11, 14, 17]. Most of this research, however, has not focussed

on learning models automatically or has focussed on models with small time lags. It would be desirable to learn a Dynamic Bayesian Network (DBN) for large datasets with large possible time lags such as the refinery time series. This is a challenging task, particularly if the DBN must be found quickly. In some applications such as analysis of oil refinery data, the explanation may be required in a very short space of time so that action can be promptly taken to prevent the escalation of abnormal events leading to possible plant shutdown.

This paper introduces a methodology that learns DBNs efficiently from MTS with large possible time lags. In section 2 we give some general background to Bayesian Networks and learning them from data. Section 3 introduces our methodology, including the representation and heuristics adopted, and the full algorithm is outlined in section 4. Other well established algorithms that exist for learning static networks are adapted to learn DBNs using our proposed representation. These are described in section 5 before being compared to one another for efficiency on synthetic datasets of varying size. Section 5 also documents extensive analysis of the efficiency and accuracy of our methodology which includes looking at how generated explanations compare to the expectations of experienced control engineers. Finally, section 6 discusses conclusions and future work.

2. BACKGROUND

2.1 Dynamic Bayesian Networks

A Bayesian network consists of the following:

(1) A set of n nodes, $\{x_1, \ldots, x_n\}$, representing the N variables in the domain and directed links between the nodes. Each node, x_i, has a finite set of r_i mutually exclusive states, v_{i1} to v_{ir_i}.

(2) To each node x_i with a set of parents, π_i there is an associated probability table, $P(x_i \mid \pi_i)$. Let w_{ij} denote the parent π_i's jth unique instantiation where there are q_i possible instantiations.

A DBN consists of the above where n nodes represent variables at differing time *slices*. Therefore links occur between nodes over different time *lags* (non-contemporaneous links) and within the same time lag (contemporaneous links).

Figure 1 shows a DBN with five variables over six time lags where each node represents a variable at a certain time slice and each link represents a conditional dependency between two nodes. Given some evidence about a set of variables at time t, we can infer the most probable explanations for the current observations.

Inference in DBNs is very similar to standard inference in static BNs. Within this paper we use a form of stochastic simulation, as described in [21], to generate the explanations in Section 5.5 through a process of scrolling and rollup. [6, 9], This involves entering observations about the state of a system at various time points, applying inference to obtain the effect of these observations on the distributions of other nodes, and then time-shifting the DBN backward so that variable 0 at time *t* becomes variable 0 at time *t-1,* variable 0 at *t-2* becomes variable 0 at *t-3,* etc. Inference can then be re-applied using the newly computed distributions as observations and the process is repeated. As this process is projected further back in time, the inferred posterior probabilities will travel further from the true states of the system. However, in some problems where a system is monitored and the time-shifting is forward in time, the current states of a system are, in fact, available from the current sensor readings. Recently, there has been much work investigating ways to remedy this "wandering" of the projected states from their true values when monitoring [3, 17, 18].

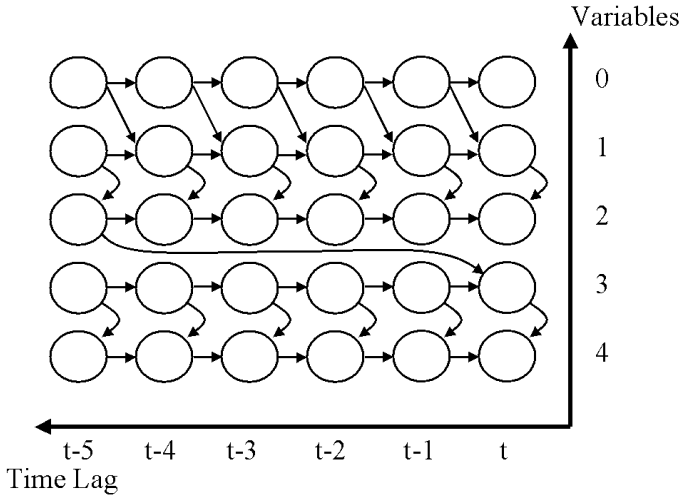

Figure 1. A DBN with five variables over six time lags

2.2 Learning Bayesian Networks

There has been a great deal of research into learning good Bayesian network structures using many different approaches such as the K2/K3 algorithms [5, 2], the Branch and Bound technique [25], and evolutionary methods [20, 29]. A good guide to the literature can be found in [4]. K2 and K3 use a greedy search which begins with an empty structure with no links. It then explores the effect of adding each of the possible links to the current structure with no links. It then explores the effect of adding each of the possible links to the current structure and the one that results in the best score is added. K2 uses this algorithm with a log likelihood metric and K3 with a description length metric, both of which are explained later in this

section. The Branch and Bound technique offers a method for performing an efficient exhaustive search by stopping any further exploration along a search path based on a bound which is calculated on the scoring metric.

When evolutionary methods are applied to *static* Bayesian networks the application of various operators is required to prevent the generation of cycles within the network. Larranaga et al. [20] used a Genetic Algorithm (GA) and applied a *repair* operator to remove cycles within the network. Wong et al. [29] used Evolutionary Programming (EP) with three operators: *freeze*, *defrost* and a *Knowledge Guided Mutation* (KGM) to improve the scalability and speed of convergence as well as remove any cycles. Using a KGM involves generating a list of all single links, ordered on their description length (DL). This list guided the mutation within an EP by adding edges which appear in the higher ranks of the list and removing edges which appear in the lower ranks. Sahami [23] used the *mutual information* between a node and its parents to select networks.

Missing data have been tackled while learning BNs in different ways. The structural EM algorithm [12] makes use of Dempster's Expectation Maximization algorithm [8] and incorporates structure search through the improvement of the likelihood of a network. Ramoni and Sebastiani use a bound and collapse method which places bounds on the final parameters of a network to account for the lack of information due to missing data [22].

We are interested in finding out how evolutionary methods can be applied to the learning of Dynamic Bayesian Networks where time is limited. For example in a control situation, given a new set of refinery data, it would be useful to be able to automatically explain certain events as quickly as possible in order to identify how the plant is being operated.

Learning BNs, both static and dynamic, involve scoring candidate network structures. In this paper we utilise two scoring metrics. Firstly, we use the log likelihood [5, 13] which is calculated using metric 1 and the higher this score the better the structure fits the dataset. Another well known metric that we use is the Description Length metric (DL) [19, 24], which has arisen out of information theory, and is constructed from the summation of the description length of a network structure (metric 2) and the description length of encoding the dataset given that model (metric 3). The lower this metric is, the better the network should fit the data.

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

$$\sum_{i=1}^{n} |\pi_i| \log(n) + \left((r_i - 1) \prod_{j \in \pi_i} r_j \right) \tag{2}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} - F_{ijk} \times \log\left(\frac{F_{ijk}}{F_{ij}} \right) \tag{3}$$

where n is the number of nodes, F_{ijk} is the frequency of occurrences in the dataset that the node x_i takes on the value v_{ik} (where there are r_i possible instantiations) and the parent nodes π_i take on the instantiation w_{ij} (where there are q_i possible instantiations) and $F_{ij} = \sum_{k=1}^{r_i} F_{ijk}$.

3. METHODOLOGY

This section is in three parts. Firstly, an assumption is made about the time series in order to develop an appropriate representation for a DBN. Secondly, we describe some heuristics which are useful in learning the structure of a DBN using genetic algorithms with the DL metric. Finally, the general methodology is outlined.

3.1 Representation

We assume that a dynamic network contains no links within the same time slice (contemporaneous links). A DBN with only non-contemporaneous links can be represented by a selection of $n = N + Q$ nodes, where N is the number of variables at a single time slice, and Q is the collection of nodes at previous time slices up to some maximum lag $MaxT$ ($Q \leq N \times MaxT$) where all 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,b,l)* where *a* is the parent variable, *b* is the child variable and *l* is the time lag. Therefore, each triple maps directly to a link in the network.

A list for $N = 5$, $MaxT = 5$ and $Q = 9$ such as *{(2,4,5), (4,3,4), (0,1,3), (2,1,2), (0,0,1), (1,1,1), (2,2,1), (3,3,1), (4,4,1)}* would represent the DBN in Figure 2. Note, as this network is for illustration only, a small number of time lags is used.

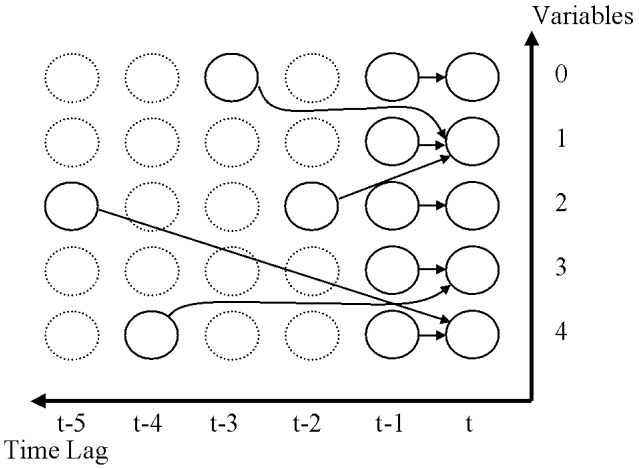

Figure 2. An example DBN using the triple representation

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. Although this assumption has already significantly reduced the search space, the number of possible network structures is still very large. For N variables and $MaxT$ possible time lags it will be $2^{MaxT \cdot N^2}$. For example, for a multivariate time series with 10 variables and a maximum possible time lag of 120, the search space will be 2^{12000}.

3.2 Useful Heuristics

In the context of learning *dynamic* networks, we can apply a standard GA [16] 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 3 is an example of the DL of a link with differing time lags using two oil refinery variables). For this reason a specific mutation has been applied to the lags of a triple (which we call *LagMutation*) and is such that each mutation is based on a uniform distribution with mean equal to the present lag.

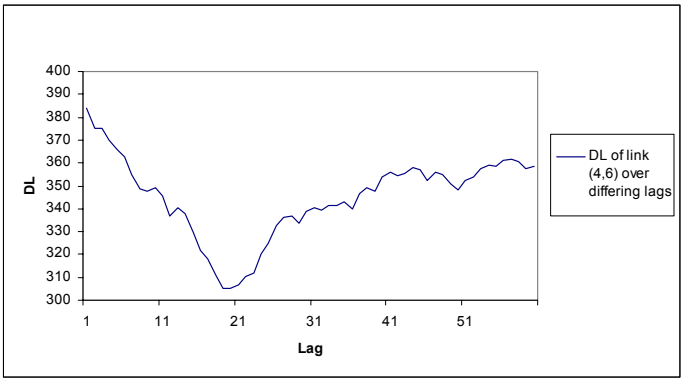

Figure 3. The DL of a single link between two oil refinery variables over 60 time lags. Note the relative smoothness of the curve.

(iii) Experimentation has shown that autoregressive links with a time lag of one (triples of the form *(a,a,1)*) are always the most common in chemical process data. This is most likely 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.

3.3 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 [20] 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. EP has been shown to be a more efficient method [26] when making use of the pre-processed single link scores through KGM. 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 [28] the entire first population with links found from the single link analysis. If 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. 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; secondly, by exploiting this knowledge in the first population by seeding it entirely with a random selection of good links.

We have found in [26] that an EP method is particularly efficient at finding a good selection of links with good correlation, particularly when we make use of self-adapting parameters (SAPs) 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. In an EP an individual could be represented as a single triple and the SAP operators can be used to exploit structure within the data to quickly find good single links. This representation would result in the entire population being the solution to the problem as opposed to the fittest individual. An EP should be better suited to this job since recombination, such as the GA's crossover, does not make much sense when chromosomes are particularly short in length, i.e. triples in this case. GAs, on the other hand, are better suited to exploring combinations of these triples. 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. The discovered list can then be used to seed an initial GA population. This is explained more fully in section 4.

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 which will maximise the recombination of the high-ranking single links.

4 ALGORITHM

Given a multivariate time series with N variables we can generate a DBN with a maximum time lag of *MaxT*. This is achieved firstly by applying Evolutionary Programming methods to a random selection of single triples in order to produce a list, *List*, of good links of length *ListSize* where the number of calls to the scoring function for the single links is limited to a pre-defined value, *c* (see steps 1-10 in the algorithm). A random selection of triples within *EPList* is then used to seed the initial generation of a GA of size *GAPopSize,* where each individual comprises a selection of triples. After *GAGenerations*, the fittest individual comprises Q triples which correspond to the DBN learned from the multivariate time series (see steps 11-19). We also use a parameter *MaxBranch* to prevent the maximum number of parents for one node becoming too high. The algorithm is described more fully below and figure 4 illustrates the general process.

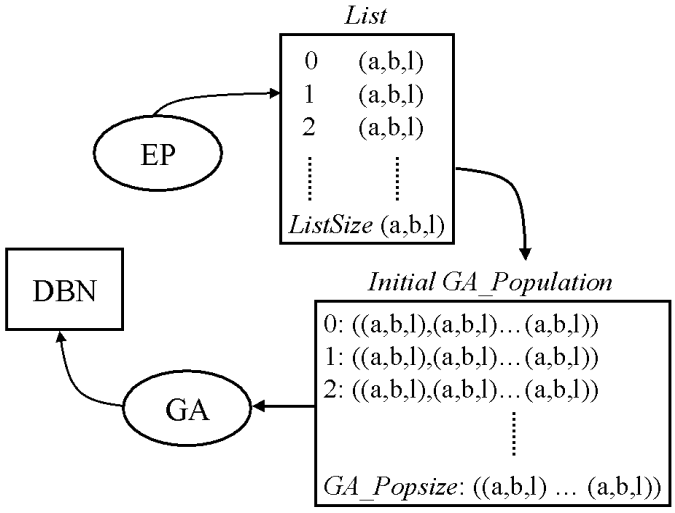

Figure 4. The Process of using an Evolutionary Program to Seed the Population of a Genetic Algorithm with a list of good scoring single triples.

EP-Seeded-GA

I/P MTS with N variables, *MaxT,*

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, *ListSize* to be the size of *List* and *c_count* to 0, *c* to be the maximum number of function calls.

3 Set the initial EP population, *List,* to a random selection of links *(a,b,l)*
where $0 \leq a < N$, $0 \leq b < N$, $1 \leq l \leq MaxT$.

4 While *c_count* < *c*

5 Score each individual triple using metric 1 or the sum of metric 2 and metric 3, and increment *c_count* for each one

6 Sort *List* 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 *ListSize.*

10 End While

11 Set *GAPopulation* to a set of *GAPopSize* valid triple lists of random length from the distribution *U(1,MaxBranch*N)* where each triple is selected from *List*.

12 Construct the network represented by each individual's triple list and set the fitness using metric 1 or the sum of metric 2 and metric 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 standard mutation to *GAMutationRate* percent of the chromosome of the offspring, essentially making a random change to those triples.

18 Add offspring to the population and remove the least fittest individuals thus reducing the population to its original size, *GAPopSize*.

19 End For

O/P The best network structure will be the one with the smallest DL / largest Log Likelihood within the last generation of *GAPopulation*.

For the proposed algorithm the following operators were used:

EPMutation Operator

EPMutation uses the notion of self adapting parameters to quickly converge to a good selection of links with good score. Within the EP, a gene can be any element of a triple *(a,b,l)*. The idea of *self-adapting parameters* [1] has been used in this context to mutate the genes using a normal distribution that is rounded up to the nearest discrete value. While this is unlikely to have any effect on mutating variables (the ordering of the variables is arbitary), it is hoped that this controlled mutation will assist the EP in 'homing in' on the best time lag for a triple, as we have found previously [26]. Here each gene, a_i, in each chromosome is given a parameter, σ_i. Mutation is defined as follows:

$$a_i' = a_i + N(0, \sigma_i) \tag{4}$$

$$\sigma_i' = \sigma_i \cdot \exp(N(0, \tau) + N(0, \tau_i)) \tag{5}$$

$$\tau = \frac{1}{\sqrt{2len}} \tag{6}$$

$$\tau_i = \frac{1}{\sqrt{2\sqrt{len}}} \tag{7}$$

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. Each σ_i is then mutated according to equation 5.

UniformCrossover Operator

Uniform crossover [27] works through the use of multiple crossover points. In the context of the DBN representation, each triple in each parent is selected to form part of one of the two offspring based on an unbiased random number generator. Therefore, each triple has a fifty percent chance of forming part of either offspring's chromosome (see Figure 5).

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-X, lag+X)*.

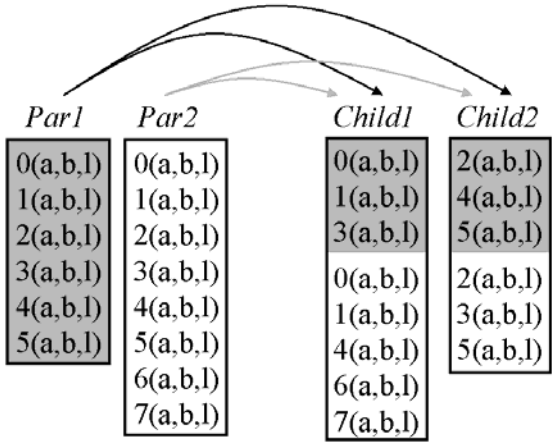

Figure 5. Uniform Crossover on the DBN Triple Representation. Any triple from either parent ahs a 50% chance of being assigned to either child.

5 EVALUATION

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

(i) Efficiency: First of all in section 5.1, we describe existing methods for learning *static* BNs which were adapted to search for DBN using our proposed representation. In section 5.2 the efficiency of these methods were compared to one another on synthetic datasets with varying *N* and *MaxT*. This involved examining the shapes of the learning curves to determine which were the best performers on larger datasets. Section 5.3 compares the best of these methods to our proposed algorithm on larger synthetic datasets and the oil refinery dataset. We also investigate how the parameters of our algorithm can affect efficiency.

(ii) Accuracy: Next, in section 5.4, the accuracy of the algorithm was 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. Finally, in section 5.5, accuracy was investigated using the oil refinery dataset through comparisons of learnt structures with causal diagrams drawn up by control engineers and feedback concerning the explanations that have been generated using the discovered networks.

5.1 Adapting Static BN Search Algorithms for DBN Search

Taking existing methods for searching for static BNs, we investigated how well they could be adapted to efficiently learn DBNs. This involved

converting the algorithms to work with the representation that we have proposed in section 3.1.

K2 / K3 (adapted for DBNs from [5] and [2])

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 based upon their time slice. It works by iterating through each node at time, *t*, 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	MTS with *N* variables, *MaxT*
1	For i=0 to *N*-1 (each node at *time t*)
2	$\pi_i = \varnothing$
3	$P_{old} = g(i, \pi_i)$
4	*Proceed* = True
5	While *Proceed*
6	Let z be the node that maximises $g(i, \pi_i \cup \{z\})$ where $z \in Q$, $z \notin \pi_i$
7	$P_{new} = g(i, \pi_i \cup \{z\})$
8	If P_{new} is a better score than P_{old} Then
9	$P_{old} = P_{new}$
10	$\pi_i = \pi_i \cup \{z\}$
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

where $g(i, \pi_i)$ is calculated using either metric 1 or the summation of metric 2 and metric 3 but only applied to the node *i*. Rather than iterating over all nodes, we only apply this to nodes at time *t*. These are the only nodes which have changing parents and so all other nodes scores will remain fixed. Step 8 depends on the metric being used (a lower score is better for DL and a higher score is better for log likelihood).

Genetic Algorithm (adapted for DBN from [20])

The Genetic Algorithm (introduced by [16]) searches for the global optimum through the application of recombination and mutation operators. These operators are applied to a population of candidate solutions which we will represent using the triple list method. The Genetic Algorithm 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. The general algorithm is as follows:

I/P	MTS with *N* variables, *MaxT*
1	Initialise random *Population* of varying length triple lists
2	Calculate the fitness of each individual using either metric with the MTS
3	For *g*=1 to *Gens*
4	Select parents from *Population* based on their fitness according to *Crossover_Rate*
5	Generate children from selected parents using *Crossover*
6	Mutate individuals using *Mutation* according to *Mutation Rate*
7	Add children to *Population* and calculate their fitness
8	Remove the least fittest individuals until *Population* is of size *Popsize*
9	End For
O/P	The fittest individual in *Population*

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, and is described below.

The Crossover Operator (see Figure 6) is defined below.

I/P	Two triple lists: *par1, par2* containing *len1* and *len2* triples respectively
1	*Cp1* and *cp2* are set to random values in the uniform distribution U(0,len1) and U(0,len2), respectively
2	Set *Child1* to the triples: $\{ par1(1,...,cp1) \cup par2(cp2,...,len2) \}$
3	Set *Child2* to the triples: $\{ par2(1,...,cp2) \cup par1(cp1,...,len1) \}$
O/P	*Child1*, *Child2*

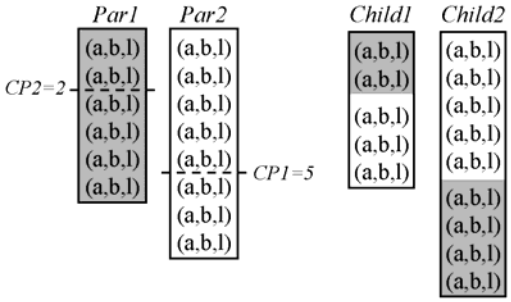

Figure 6. The Crossover Operation Applied to two Parent Triple Lists to generate two new Children Triple Lists. Crossover points were 2 and 5 for *Par1* and *Par2* respectively.

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:

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

Fitness is calculated using either of the scoring metrics. As we wish to increase fitness with each successive generation, the DL is inverted whenever a comparison is made (steps 4 and 8 in the algorithm).

Evolutionary Program (adapted for DBN from [29])

The Evolutionary Program (EP) 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. An EP is similar to a GA in that it uses a population of chromosomes whose fitness we try to improve through mutation. However, in EP there is no recombination.

Knowledge Guided Mutation (*KGM*) 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. This means that a pre-processing stage is required in the same way as our methodology. However, rather than using an approximate pre-processing method, Wong used an exhaustive search over all single links. For our experiments, the log likelihood of a single link was used to bias the mutation where the log likelihood metric was used. KGM takes a list, *List*, of all possible links (triples) in the DBN which have been scored according to Log Likelihood or Description Length. Given a parent, it then randomly adds or deletes a triple. A triple is more likely to be added if it has a better score and is more likely to be deleted if it has a worse

score. Mutation is identical to the Mutation operator applied to the GA and once again, as in the GA, the DL is inverted whenever a comparison is made as we are trying to maximise the fitness.

```
I/P     MTS with N variables, Pre-processed List of all scored single links
1       Initialise random Population
2       Calculate the fitness of each individual using either metric with the
        MTS
3       For g=1 to Gens
4           Generate a child for each member of Population using KGM
            and List
5           Add each child to Population and
9           Randomly Mutate all individuals in new Population and
            score their fitness
10          Select the Popsize individuals with the highest scores to
            recreate the next Population
11      End For
O/P     The Fittest Individual in Population
```

5.2 Comparing Efficiency of Adapted Methods on Synthetic Data

The algorithms were tested on various synthetic datasets generated from DBNs. Each DBN consisted of differing numbers of variables and time. The generated data contained 1000 data points for each Boolean variable. We used a stochastic simulation inference scheme to generate the datasets. This involved starting with random values for all nodes at time $< t$, and using these values as observations to the inference algorithm. New values could then be assigned to variables at time t based on the computed posterior distributions. By time-shifting the DBN forward one time slice, each successive time point in the MTS could be assigned values based on the previous values. For more information on stochastic simulation as inference in Bayesian networks see [21].

Figure 7 shows the learning curves of the search methods on synthetic DBN data of varying N and $MaxT$ using the description length metric. The y-axis represents description length and the x-axis represents the number of Function Calls (FC). Figure 8 shows the results of exactly the same experiments carried out using the log likelihood scoring metric on the same set of synthetic datasets. Notice that the GA starts off on a large number of FC. This is due to the best population size being found was 100 resulting in many FC required for the initial population. EP on the other hand is found to have optimal performance when its initial population (of size 10) contains no links, and K"/K3 does not use a population of candidates.

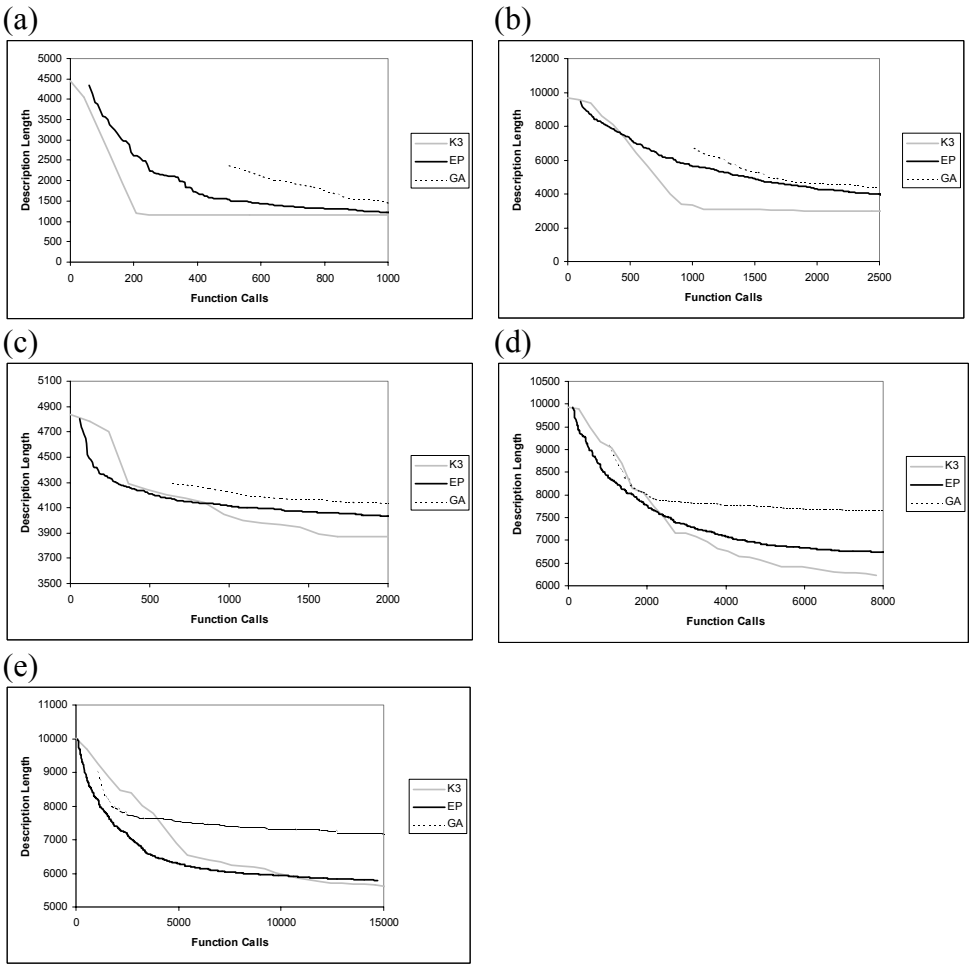

Figure 7. Comparing the Search Methods on DBN-Generated MTS using Minimum Description Length (a) *N*=5, *MaxT*=10; (b) *N*=10, *MaxT* =10; (c) *N* =5, *MaxT* =30; (d) *N* =10, *MaxT* =30; (e) *N* =10, *MaxT* =60

It is evident from graphs 7(a) and 8(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. Notice that as either *N* or *MaxT* increases, graphs 7(b), (c) and (d), and 8(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 7(e)

and 8(e). This is likely to be due to the efficient recombination of good links in the first few generations followed by the reliance upon chance mutation to find any further links. In short, on datasets with larger dimensionality and larger maximum time lag, EP performs more efficiently than any of the other methods. We will now see how our proposed methodology compares to EP on synthetic and real-world datasets with larger *N* and *MaxT*.

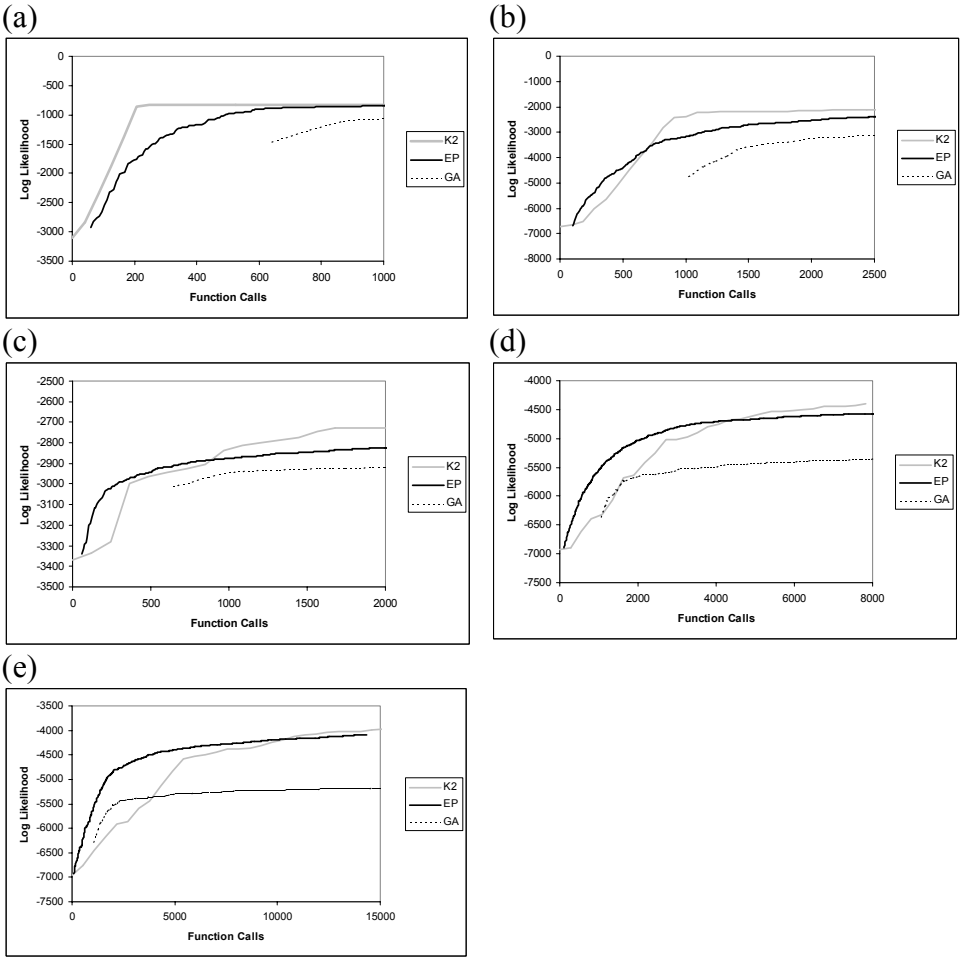

Figure 8. Comparing the Search Methods on DBN-Generated MTS using Maximum Log Likelihood (a) *N*=5, *MaxT*=10; (b) *N*=10, *MaxT* =10; (c) *N* =5, *MaxT* =30; (d) *N* =10, *MaxT* =30; (e) *N* =10, *MaxT* =60.

5.3 Comparing Efficiency of EP-Seeded GA with Standard EP on Larger Synthetic and Real World Datasets

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 parametrised as shown in the table 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.

Figures 9a to 9f 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 higher 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 10 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%.

	c	*PopSize*	*GAPop-Size*	*ListSize*	*GACross-overRate*	*GAMutat-ionRate*	*LagMutat-ionRate*
Standard EP	100%	10	-	100%	-	-	-
EP-Seeded GA	20 / 100%	-	10	2.5%	0.8	0.1	0.1

Table 1. Parameters for the Standard EP and EP-Seeded GA

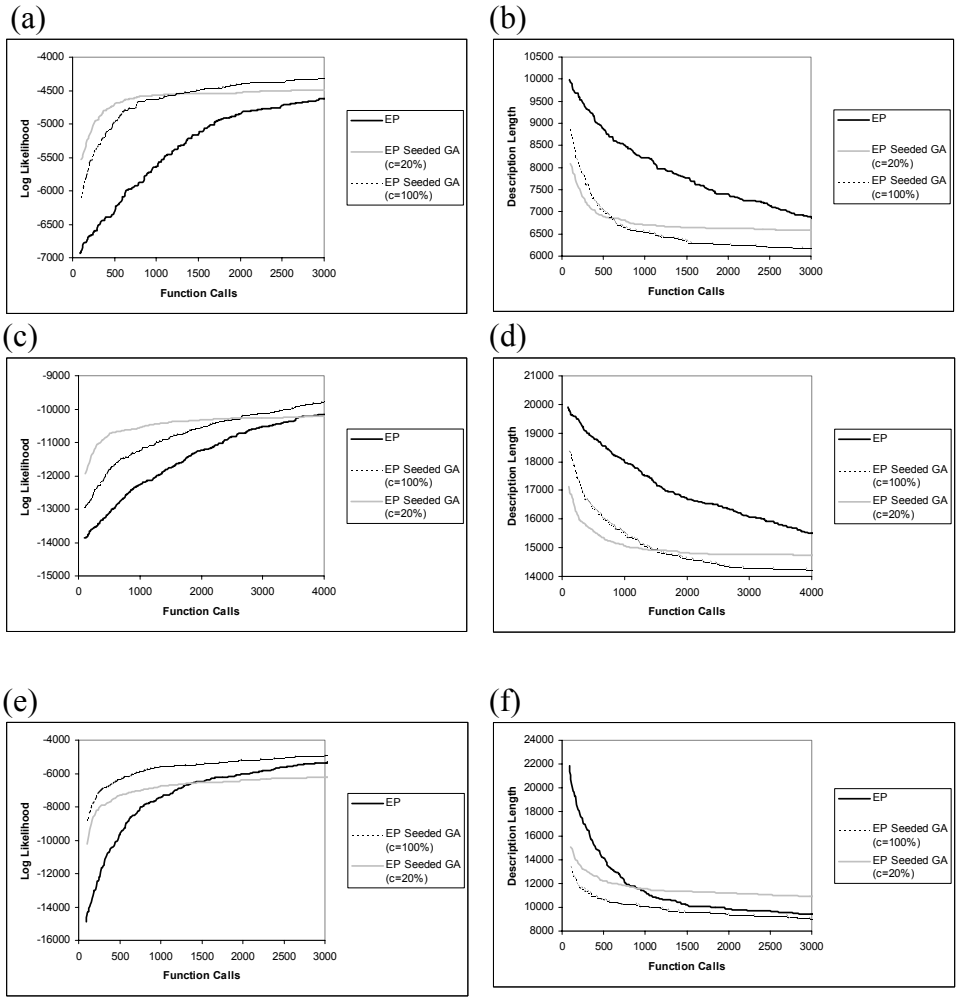

Figure 9. Performance on Synthetic Datasets: (a) (b) *N=10, MaxT=60;* (c) (d) *N=20, MaxT=60*; and Oil Refinery Datasets: (e) (f) *N=11, MaxT=60.*

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.

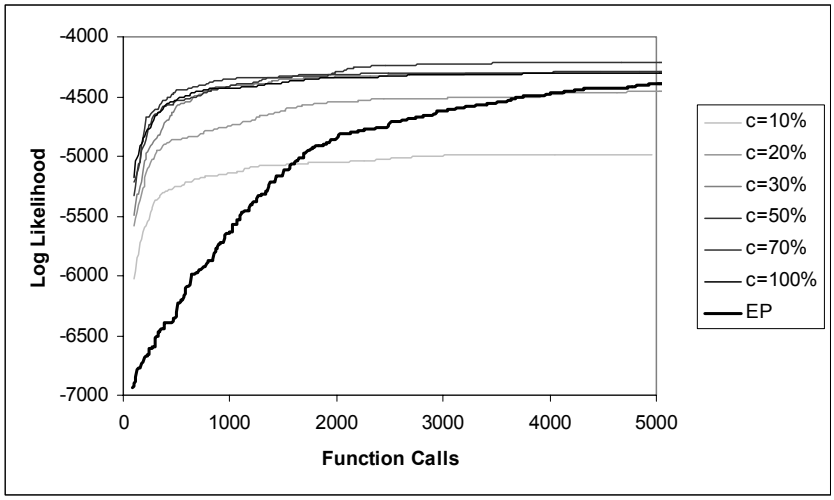

Figure 10. Performance on Synthetic Dataset with N=10 and $MaxT$=60 with varying number of calls, c, in the EP-Seeding Stage of EP-Seeded-GA.

5.4 Structural Comparison of DBNs learnt using EP-Seeded GA and Standard EP

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 the links that were found within the resulting structures but should not have been due to spurious correlations and implicit dependencies, 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 less, then it is not deemed to be different from the original network. This value was arrived at as it was decided that an explanation that was incorrect in the time aspect by three minutes or less would not affect the overall quality. Anything larger than three minutes, however, may be misleading to a control engineer using the tool. 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. We only show results from this analysis using the log likelihood metric. DL scores produce almost identical results.

The SD analysis (Table 2) shows a surprising result in the standard EP algorithm. Whilst the SD of the EP-Seeded-GA steadily decreases as function calls increase, the standard EP actually increases in SD for some time before falling. 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. A

spurious correlation is a dependency that appears to exist between two nodes due to a common parent between the two nodes (see figure 11a), an implicit dependency is one which appears to exist due to indirect causes (see figure 11b).

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 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.

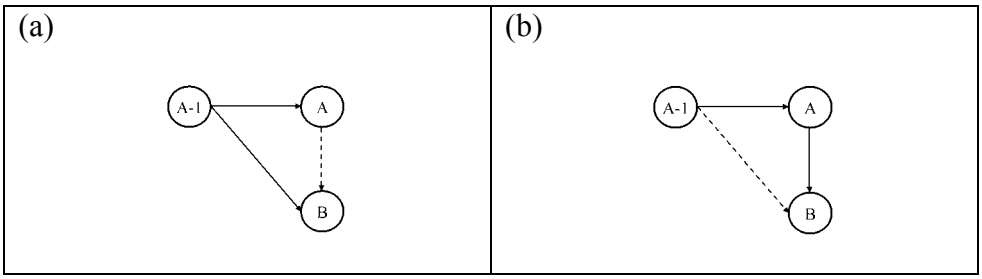

Figure 11. (a) Spurious Correlation denoted by a dotted line between node A and node B. (b) Implicit Dependency between node A-1 and node B.

The hypothesis that spurious correlations account for the rise in SD for standard EP is supported by figure 12b. 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 correlations actually increases. It appears that the EP-Seeded-GA avoids this growth in spurious correlations with all elements of SD generally decreasing (see figure 12a).

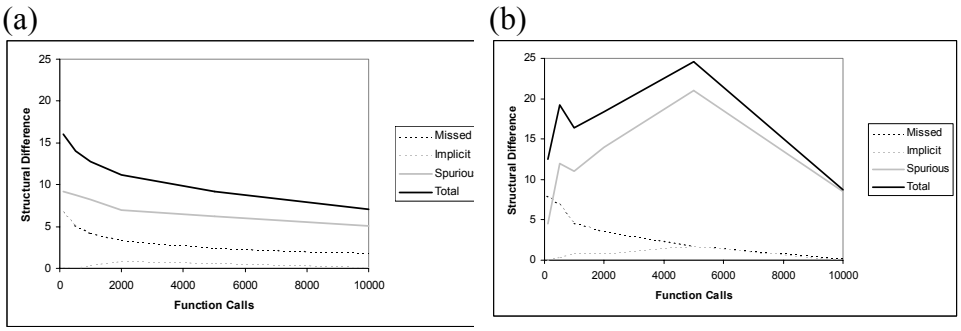

Figure 12. Breakdown of Structural Difference on Synthetic Dataset where *N=10* and *MaxT=60* using (a) EP-Seeded-GA and (b) Standard EP.

5.5 Quality of Explanations

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 the variables in the oil refinery MTS (see Figure 14). 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 13 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%, and the algorithm was stopped after 1500 and 2000 function calls for the 10 variable and 20 variable dataset, respectively. Shaded boxes in figure 13 represent observed nodes.

It is very encouraging to note that the current algorithm detects all of the relationships within figure 14 correctly except for those limited by *MaxBranch* being set to 3. For example, the explanation in figure 13a has captured tail gas flow (TGF) as being dependant upon sponge oil flow (SOF and top temperature (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 within the dataset. From figure 13b we can see that bottom product flow (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 13c), we see an increase in the probability of other variables being the cause of BPF's current state.

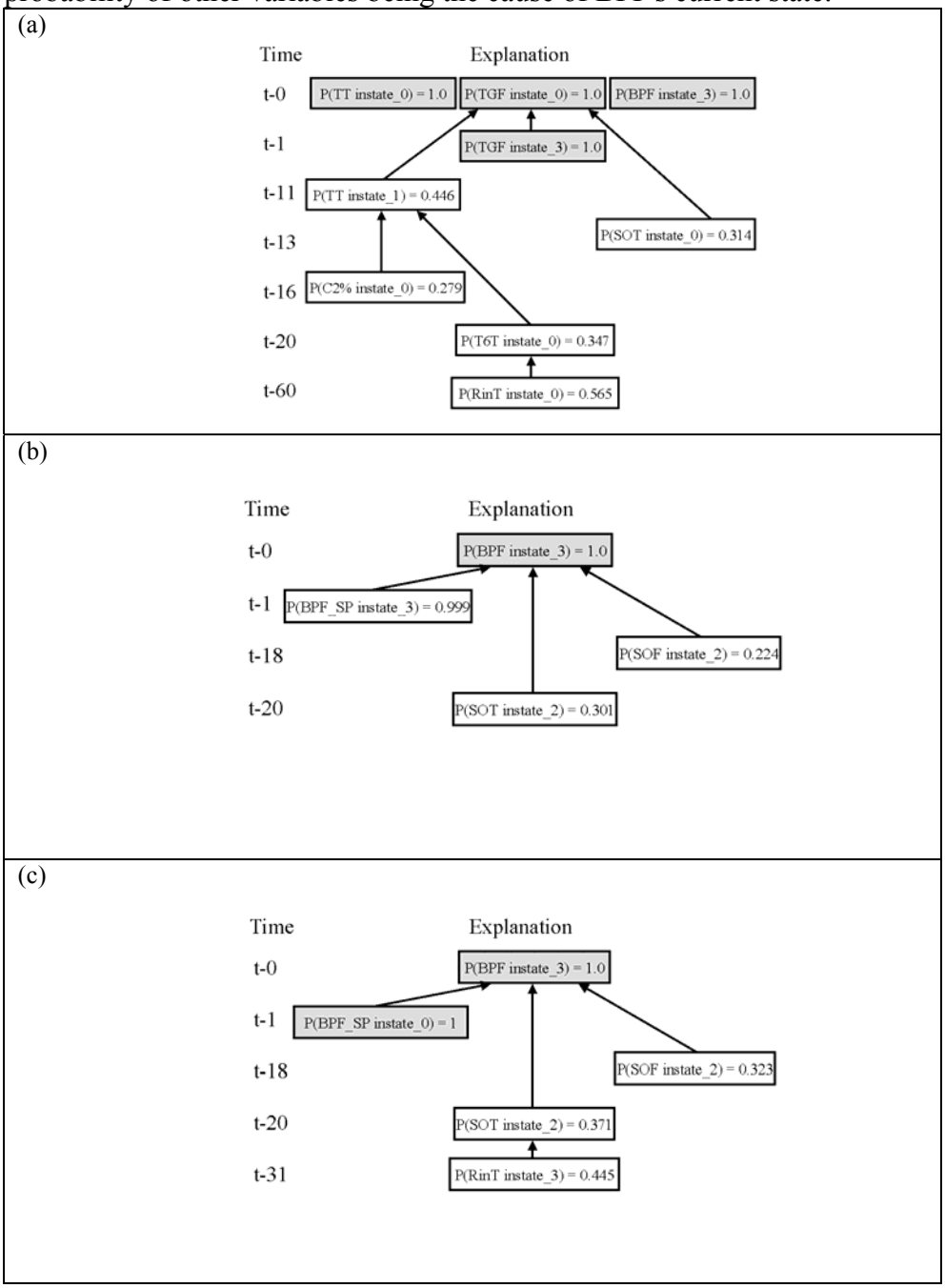

Figure 13 - Sample explanations generated using the refinery data. Shaded blocks represent observed variables.

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 CCF between two variables (see figure 3) where a strong dependency from variable 4 to variable 6 will 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 14 if feed flow (FF) affects TGF and BPF but is missed out of the dataset, it is quite possible that a dependency will be found between TGF and BPF. It is, therefore, important to ensure that all of the relevant variables are included within the dataset and a good discretisation policy is adopted.

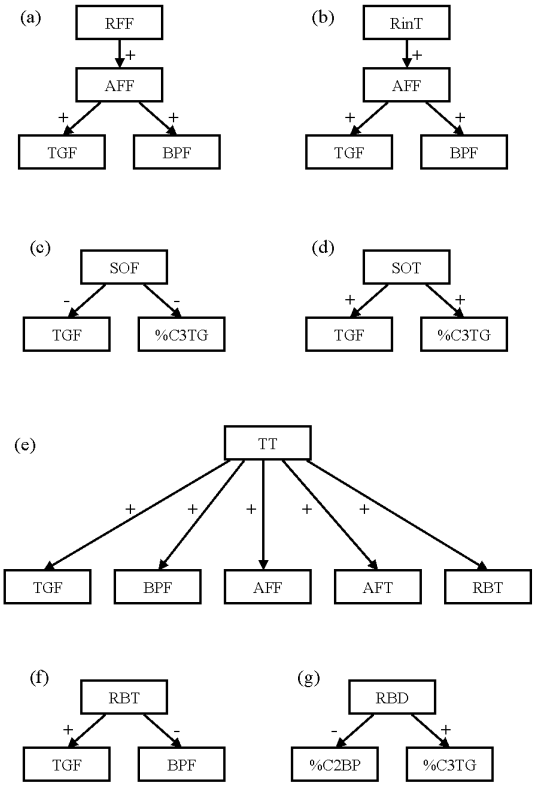

Figure 14 - Example Dependency Diagrams constructed from advice of Control Engineer

7. 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). It is a very challenging task and we have investigated the use of

evolutionary methods in achieving this. 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 some success in managing this complexity, especially in comparison to algorithms that exist for static networks and which we have adapted for dynamic models. When good quality explanations must be produced in as short amount of time as possible, our EP-Seeded GA is far more efficient than any other methods and the accuracy of explanations generated from oil refinery datasets are very encouraging.

Future research will involve exploring ways to improve the accuracy of this algorithm through better handling of discretisation (such as [10]) and parameterisation, and ways to extend this sort of algorithm to more complex data such as those multivariate time series where dependencies can change over time.

The authors would like to thank their sponsors: The Engineering and Physical Science Research Council, UK; Honeywell Hi-Spec Solutions, UK and Honeywell Technology Centre, USA. We would also like to thank BP Oil for supplying the dataset.

References

[1] T. Baeck, "Evolutionary Algorithms: Theory and Practice", Oxford University Press, 1996.

[2] 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.

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

[4] W. Buntine, "Theory Refinement on Bayesian Networks", Uncertainty in AI, pp. 52-60, 1991.

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

[6] P. Dagum, A.Galper, E. Horvitz, "Dynamic Network Models for Forecasting", Uncertainty in AI, pp 41-48, 1992.

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

[8] A.P. Dempster, N.M. Laird, D.B. Rubin, "Maximum-Likelihood from Incomplete Data Via the EM Algorithm", J. Royal Statist Soc Ser B 39, pp. 1-38, 1977.

[9] J. Forbes, "The BATMobile: Towards a Bayesian Automated Taxi", Reasoning Under Uncertainty, pp. 1878-1885, 1999.

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

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

[12] N. Friedman, “The Bayesian Structural EM Algorithm”, 14th Annual Conference on Uncertainty in AI, pp. 129-138, 1998.

[13] D. Geiger, “An Entropy Based Learning Algorithm of Bayesian Conditional Trees”, Proceedings of the 8th Conference on Uncertainty in AI, pp 92-97, 1992.

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

[15] D. Heckerman, “Bayesian Networks for Data Mining”, Data Mining and Knowledge Discovery, Vol 1, pp 79-119, 1997.

[16] J.H. Holland, “Adaptation in Natural and Artificial Systems", The University of Michigan Press, 1975.

[17] K. Kanazawa, D. Koller, S. Russell, “Stochastic simulation algorithms for dynamic probabilistic networks”, Proceedings of the 11th Conference on Uncertainty in AI, pp 346-351, 1995.

[18] D. Koller, “Approximate Probabilistic Inference in Dynamic Processes”, Working Notes of the 1996 AAAI Spring Symposium on Learning Dynamic Systems, 1996.

[19] W. Lam, F. Bachus, “Learning Bayesian Networks. An approach based on the MDL principal”, Computational Intelligence 10(3), pp 269-293, 1994.

[20] P. Larranaga, M. Poza, Y. Yurramendi, R. Murga, C. Kuijpers, “Structure Learning of Bayesian Networks using GAs”, IEEE Trans. Pattern Analysis and Machine Intelligence, vol 18, no.9, pp 912-926, 1996.

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

[22] M. Ramoni, P. Sebastiani, “The Use of Exogenous Knowledge to Learn Bayesian Networks from Incomplete Databases”, Proceedings of Intelligent Data Analysis 97, LNCS 1280, Springer-Verlag, pp. 537-548, 1997.

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

[24] J. Suzuki, “A Construction of Bayesian Networks from Databases Based on the MDL Principle”, Proceedings of the 9th Conference on Uncertainty in AI, pp. 266-273, 1993.

[25] J. Suzuki, “On an Efficient MDL Learning Procedure Using Branch and Bound Technique”, Technical Report COMP95-27, Institute of Electronics, Information and Communication Engineers, 1995.

[26] 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.

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

[28] 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, Orlando, 1999.

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