Extending Evolutionary Programming Methods to the Learning of Dynamic Bayesian Networks

Allan Tucker & Xiaohui Liu
Department of Computer Science,
Birkbeck College, University of London,
Malet Street, London WC1E 7HX, UK.
[atuck06, hui] @dcs.bbk.ac.uk
+44 171 631 6723 / 6711

Abstract

Recent work has shown that for finding *static* Bayesian network structures, an Evolutionary Programming (EP) approach that exploits the description length of single links is better suited than a standard Genetic Algorithm (GA). We extend this work to find good *dynamic* Bayesian network structures that can have large time lags. We do this through the use of a new representation of dynamic Bayesian networks for EPs and a new operator, *swap*, designed specifically with a dynamic Bayesian network in mind. In this paper Lam's knowledge guided operator for static networks is compared with the new *swap* operator when both are used in conjunction with the new representation. Experiments are carried out on synthetic datasets and a real world oil refinery process time series. The results indicate that the new operator is better suited to finding good structures in a shorter amount of time.

1 Introduction

Many complex chemical processes record multivariate time series data every minute. This data will be characterized by a large number of interdependent variables whilst some variables may have no substantial impact on any others. There can be large time lags between causes and effects (over 120 minutes in some chemical processes). If we want to diagnose a particular event within the system automatically, we would first have to learn the dependencies between these variables that are evident within the data. Diagnosis, in these situations, will want to be performed as close to real time as possible and so an approximate algorithm to learn the dependency structure and perform inference must be found. This paper presents an automatic method to do this in order to perform the diagnosis of particular events. This makes use of a paradigm known as the dynamic Bayesian network (DBN) which is learnt using an EP approach. We combine a new representation for a DBN with a proposed *swap* operator designed, specifically, to speed the convergence when learning dynamic Bayesian networks.

In the next section we introduce the Dynamic Bayesian Network as a tool for diagnosis and describe various methods for learning static Bayesian networks from data, including recent work on the use of evolutionary and genetic methods, in particular the MDLEP method [Lam 1998]. This makes use of the knowledge guided mutation operator which we analyse at the end of section 2. To the best of our knowledge, no research has been carried out on using evolutionary methods to learn DBNs or, indeed, any methods for learning DBNs with large time lags. In section 3 we propose a way of doing this by introducing a representation of a dynamic Bayesian network for EP. We then describe the proposed algorithms and the associated operators. Section 4 presents a comparison of the performances of the operators on a collection of synthetic datasets and a real world oil refinery time series. Operators with the fastest convergence rate will be better suited to problems such as those posed by diagnosis within process datasets where speed is essential.

2 EPs and GAs for Bayesian Networks

A Bayesian Network (BN) is a paradigm for modelling a system using probabilities [Neapolitan 1989; Pearl 1988]. It is a way of storing the joint probability distribution for all variables in a domain by exploiting conditional dependencies between them. It consists of a directed acyclic graph (DAG) and a set of conditional probability distributions. The DAG is made up of *nodes* which represent variables in the domain and directed *links* between them which denote a conditional dependence between them. Each node has a probability distribution over each of its *states*, which is conditioned upon its parents. The set of parents of a node are those with a link pointing to it. By setting the probabilities of some nodes' states equal to 1 we can enter observations about the system and by using certain algorithms we

can perform inference in the network in order to reason about the system.

Dynamic Bayesian Networks (DBNs) are an extension to their static counterpart in that they model a system over time [Dagum 1995; Friedman 1998; Ghahramani 1998; Kanazawa 1995]. A node represents a particular variable at a particular *time slice*. Links in DBNs occur between nodes within a time slice (contemporaneous) and over different time slices (non-contemporaneous). See Figure 1 for an example DBN over six time slices with four variables.

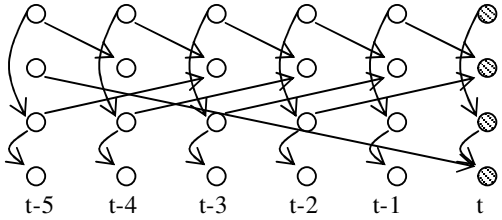

Figure 1. A DBN with four variables over six time slices. Hashed nodes are evidence, E_t

Given some evidence about a set of variables at time t, E_t, we can infer the *posterior* distributions of variable states at differing time lags (l), X_{t-l}. This can be performed using various algorithms, stochastic simulation being the most commonly used when dealing with larger networks (see [Pearl 1988]. This is an approximate method as exact inference has been shown to be an NP hard problem [Cooper 1990]. Stochastic simulation works by repeatedly generating states for each variable in the DBN according to their probability distributions given any observations entered into the network. The method has been shown to converge to a close approximation of the true posterior probability distributions. Therefore, given the DBN structure and conditional probability distributions (which can be learnt relatively easily from a dataset with no missing values), we can infer the posterior distributions in order to automatically diagnose a particular set of observations.

There are various methods for scoring the structure of either a dynamic or static BN according to how well it represents a dataset : Bayesian Methods [Cooper 1991], Minimum Description Length (MDL) [Lam 1994], Maximum Likelihood (ML) [Geiger 1992]. A detailed guide to the literature can be found in [Heckerman 1996]. The Minimum Description Length is of interest because it penalises largely connected networks which would be undesirable for efficient inference. The Description Length (DL) can be calculated by summing equations 1 and 2 below:

$$DL_{Model} = \sum_{n_i \in n} [k_i \log_2(n) + d(s_i - 1) \prod_{j \in F_{n_i}} s_j] \quad (1)$$

$$DL_{Data} = \sum_{n_i \in n} [\sum_{n_i, F_{n_i}} M(n_i, F_{n_i}) \log_2 \frac{M(F_{n_i})}{M(n_i, F_{n_i})}] \quad (2)$$

where n is the number of nodes; for node n_i, k_i is the number of its parents, F_{n_i} is its set of parents, s_i is the number of states it can be in and s_j is the number of values a particular variable in F_{n_i} can take on; d is the number of bits needed to store a numerical value; $M(.)$ is the number of times a particular instantiation exists in the database This is known as the *mutual information.* Equation 1 is the DL of the model (DL_{model}) and equation 2 is the DL of the data given the model (DL_{data}). The lower the DL of a particular network, the better the model.

Once a scoring metric is decided upon some method of search must be chosen in order to quickly find DBNs with a good score (a low DL). Evolutionary methods have recently been employed on static networks to find global solutions quickly and seem to show promising results. When these methods are applied to the networks the application of various operators is required in order to prevent the evolution of cyclic networks. This is because the operators used are not closed operators. In [Larranaga 1996] a Genetic Algorithm is used and a 'repair' operator is applied to remove cycles. In [Lam 1998] evolutionary programming is used with three operators: freeze, defrost and a knowledge guided mutation (KGM) operator. These are used to improve the scalability and speed of convergence and ensure any links that generate cycles are removed. The KGM operator works by calculating the DL of all possible single links beforehand. This information is then used in the mutation of individuals so that links with a low DL are more likely to be added and links with high DL are more likely to be removed. This assumes that the DL of a single link implies something about its effect on a global network. By examining some real world data and some datasets generated from a number of small Bayesian networks we have found that this is the case.

Therefore, for each Bayesian network we can construct a list of all the possible single links and order them upon their DL using the data generated. Then we can see where the original links of the network lie on this list. Figure 2 shows the positioning of the real links (diamonds) within the ordered list for one such network. It can be seen that the majority of them lie in the higher rankings. This suggests that the DL for a single link

may be a good heuristic as to whether it is part of the global network. In the next section we exploit this generated list in full whilst learning DBNs.

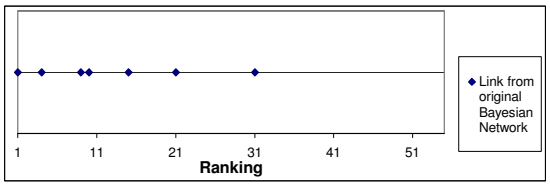

Figure 2. The ranking of the true single links in a generated dataset from a Bayesian network.

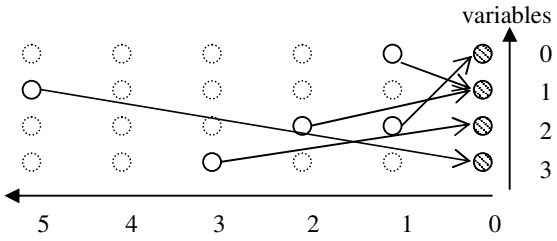

Figure 3. A DBN with no contemporaneous links. Hashed nodes are observed states.

3 Method

Representing a DBN for EP

A DBN with only non-contemporaneous links is represented by a selection of $N + P$ nodes, where N is the number of variables and represents each variable at time slice t, and P is the collection of nodes representing variables at previous time slices up to some maximum lag $maxl$. Note that $P \leq N \times maxl$. We can use a list of triples, (a,b,l) , to represent a possible network 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. So a list for N=4 and with P = 5 such as (0,1,1), (1,3,5), (2,1,2), (2,0,1), (3,2,3) would represent the DBN in Figure 3.

EP to learn single link knowledge for seeding initial population

[Lam 1998] use the DL of a single link to guide their mutation. However, their method only makes use of the heuristic once for each individual every generation. If we want to exploit this knowledge as soon as possible in order to find better networks in fewer generations we can seed the entire first population with links found from the single link analysis. What is more, we can construct a list of these single links more rapidly by using an EP. If speed is of the essence and we need a good network in as short a time as possible we can speed up the algorithm in two ways: firstly, by using an approximate method to find a good list of single links rather than an ordering of 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 that EP is particularly efficient at finding a good selection of links with low DL, particularly when we make use of self-adapting parameters. We, therefore, experiment with the use of EP to find a good ordered list of links with low DL and avoid having to explore every possible single link.

If the initial population contains links with low DLs as found using an EP, it would be useful if the next stage of search emphasised the recombination of these links. For this reason we have developed a new operator, *swap*, which will maximise the recombination of the high-ranking DL links. What is more, if we look at how the DL of a triple varies with differing lags we find that it is a relatively smooth curve (see Figure 4 for an example of the DL of a link with differing time lags). For this reason we have designed a specific 'sliding' mutation for *swap* where each mutation is only applied to the lags of a triple and is such that each mutation is an addition or subtraction to the previous value of the lag.

Early experimentation showed that autoregressive links (triples where a=b) with a time lag of one were always the most common in chemical process data. For this reason, these links were excluded from possible triples and automatically inserted into the networks before evaluation.

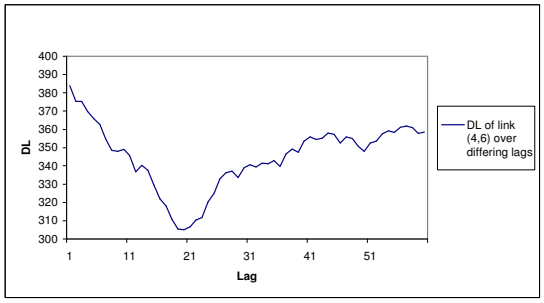

Figure 4. The DL of a single link with varying time lag. Note the smoothness of the graph.

We can now describe the two algorithms in full. Algorithm 1 uses the KGM operator and Algorithm 2 uses *swap* on a population that is seeded with links of low DL.

Algorithm 1

1. Given a multivariate time series, discretize any variables that are continuous.
2. Generate a list of all single links ordered on their DL (summing equations 1 and 2).
3. Set the initial population to a random selection of *P* triple-lists where for each triple, *(a,b,l),* $0 \le a < N$, $0 \le b < N$, $1 \le l \le maxl$, $a \ne b$.
4. Generate the DBN represented by each triple-list.
5. Calculate the DL of each DBN.
6. For i = 1 to *Generations* DO
   - Apply *KGM Operator* to two random parents in the top *OpRate* of the population in order to generate offspring
   - Add all valid offspring to the population and remove the least fittest individuals thus reducing the population to its original size, *PopulationSize*
7. The best network structure will be the network with the smallest DL within the last generation.

Algorithm 2

1. Given a multivariate time series, discretize any variables that are continuous.
2. Set *ListSize* equal to *Limit%* of all possible single links.
3. Set the initial population of *List* of size *ListSize* to a random selection of single triples *(a,b,l)* where $0 \le a < N$, $0 \le b < N$, $1 \le l \le maxl$, $a \ne b$.
4. For i = 1 to *ListGenerations* DO
   - Make a copy of each triple and apply *Preprocess_Op* to each duplicate
   - Add the mutated duplicates back to the population
   - Remove the lower ranking links until the population is back to its original size, *ListSize*
5. Set the initial population in the main algorithm to a random selection of *P* triple-lists from *List*.
6. Generate the DBN represented by each triple-list.
7. Calculate the DL of each DBN.
8. For i = 1 to *Generations* DO
   - Apply *Swap Operator* to generate offspring. Different forms of this are described below
   - Add all valid offspring to the population and remove the least fittest individuals thus reducing the population to its original size, *PopulationSize*
9. The best network structure will be the network with the smallest DL within the last generation.

Preprocess_Op:

This EP operator uses the notion of self-adapting parameters. Each gene, x_i, is given a parameter, σ_i. This is used as the standard deviation of a normal distribution with which mutation is applied to x_i (equation 3). The standard deviations, themselves, are updated according to equations 4 where s is calculated from equation 5 for each chromosome and s_i is calculated from equation 6 for each gene (taken from [Baeck 1996]) where n is 3, i.e. the size of a triple:

$$x_i' = x_i + N(0, \sigma_i) \tag{3}$$

$$\sigma_i' = \sigma_i \cdot \exp(s + s_i) \tag{4}$$

$$s = N(0, \frac{1}{\sqrt{2n}}) \tag{5}$$

$$s_i = N(0, \frac{1}{\sqrt{2\sqrt{n}}}) \tag{6}$$

1) *KGM Operator*

1. Choose two random individuals within the top *OpRate* × *PopulationSize* fittest individuals and clone them.
2. For each clone do the following:
   - Add a new link from the generated list with a high probability of having a low DL
   - Remove a random link from the network with a higher probability of deleting one with high DL
3. Randomly mutate one triple.

2) *Swap Operator*

1. Choose two random parents, par_1, par_2, within the top *OpRate* × *PopulationSize* fittest individuals.
2. Set the first offspring equal to par_1 and the other to par_2.
3. For the first offspring, exchange each triple with the other offspring's according to a random number generator with a probability of 50%.
4. For a random 10% of each individual's triples DO
   - Add or subtract a random value up to a maximum of 10% of the *maxl*, to the lag. This is the sliding mutation

4 Results

The algorithms are evaluated in the following two ways. Firstly a set of experiments were carried out involving synthetic datasets generated from a DBN. The two operators were then tested for the number of

generations until the original structure was found. Secondly a real world oil refinery process dataset was used to find the structure with the lowest DL.

Synthetic Data

Firstly, a synthetic dataset was produced from a DBN with maximum lag being set at 30 minutes. See an example in Figure 5. The performance of the operators were compared and the number of generations taken to converge to the correct solution was recorded. This experiment was repeated with 10 randomly generated network structures. For the algorithms in section 3 the following parameter values were used: *Generations* = 500, *PopulationSize* = 20, *OpRate* = 80%, *MutationRate* = 1%, *Limit* = 25, *P* = 5, and *ListGenerations* = 30.

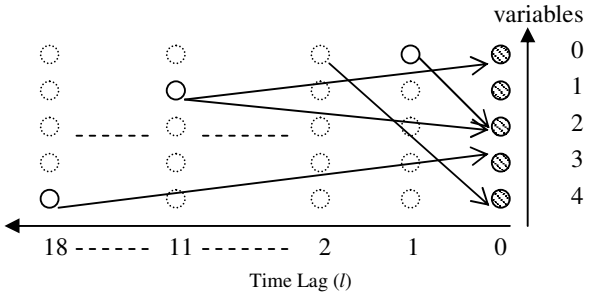

Figure 5. An example DBN used to generate the synthetic data. Maximum lag to search for was set to 30 minutes.

Figure 6 shows the performance of the two operators on one of the synthetic datasets. Both methods converged rapidly to a near solution. However, the *swap* operator performed better reaching the correct solution after 79 generations whilst the KGM took 190. It should be noted that KGM was very close to the optimal after 81 generations but took a long time to find the correct time lags.

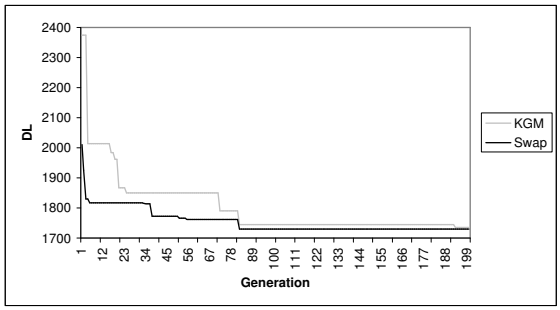

Figure 6. The Performance of the KGM and *swap* operators on a synthetic dataset.

Table 1 shows the average number of generations to find the original DBN over all ten datasets for both operators. It can be seen that the *swap* operator is consistently faster at converging to the correct structure with a significantly lower average.

Algorithm / Operator used	Average Number of Generations
Algorithm1 / Knowledge Guided Mutation	217.1
Algorithm2 / *swap*	96.6

Table 1. The average number of generations to find the correct original DBN structure for KGM and *swap*.

Oil Refinery Data

Figure 7 compares the performance of the different operators when applied to the multivariate oil refinery time series. This was a multivariate time series with 11 variables and 1000 minutes of data. All continuous variables had been previously discretized. If we set the maximum time lag to 60 minutes and exclude any autoregressive links then the number of possible network structures will be $2^{11\times10\times60}$. To find a good structure from this huge number of candidates is a very challenging task.

For the algorithms in section 3 the following parameter values were used: *Generations* = 2000, *PopulationSize* = 30, *OpRate* = 80%, *MutationRate* = 1%. *Limit* = 25, *P* = 15 and *ListGenerations* = 50.

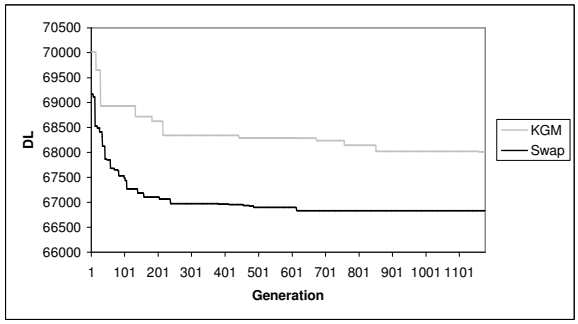

Figure 7. Comparison of the operators *swap* and KGM on the oil refinery dataset.

Once again it is apparent that the fastest convergence was obtained through the use of the *swap* operator. It started off with a lower DL and continued to improve at a faster rate than KGM. The *swap* operator had the head start, probably due to the seeding of the initial population; maximal recombination and slide mutation then ensured a fast convergence to a good structure. The structure found using the *swap* operator after 2000 generations is shown in Figure 8. This corresponded to the best triple-list in the final population : { (0,6,1), (1,10,53), (1,8,52), (1,3,16), (1,6,1), (2,0,1), (5,9,1), (7,9,24), (7,3,15), (7,2,10), (10,1,59), (10,4,52), (10,5,36), (10,7,15), (10,8,1) }.

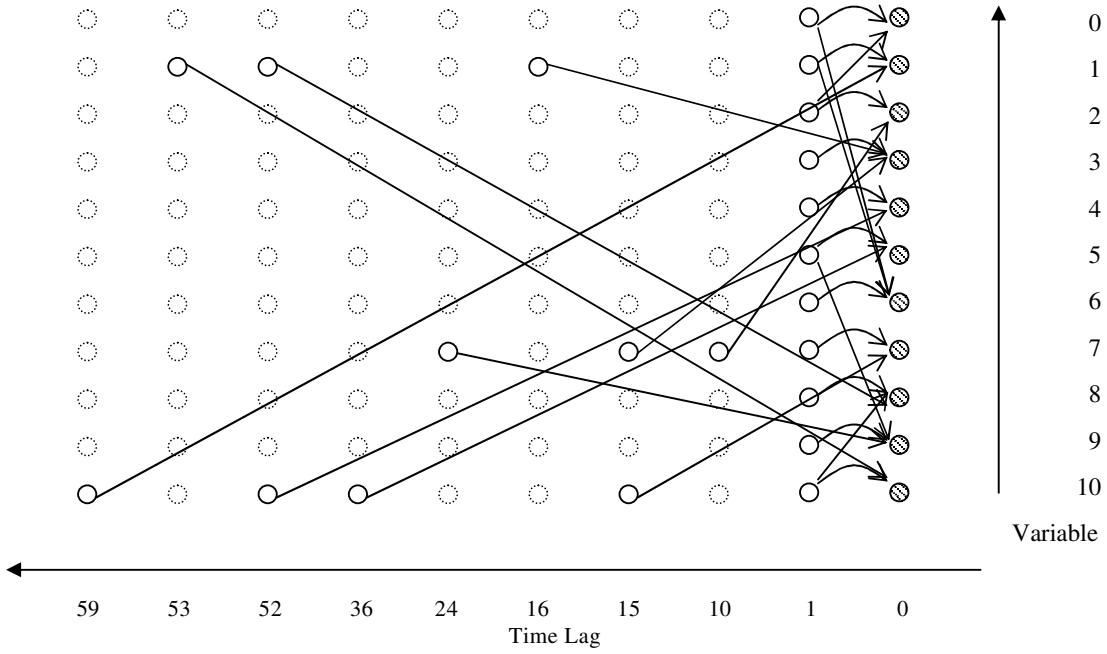

Figure 8. The Network learnt using *swap* after 2000 generations with the autoregressive links added.

The network structure found after 2000 generations contained many links which were expected from the refinery dataset. Those being between variables that were known to be related such as Sponge Oil Temperature affecting Top Temperature with a delay of approximately 5 minutes. However, other unexpected links were found from likely effects to likely causes. This could have been due to a large dependency from one variable to another resulting in a high mutual information between the variables in the opposite direction. A few other unexpected links occurred where the data was shown to exhibit little variation.

5 Conclusions and Future work

The learning of dynamic Bayesian networks with large time lags has great potential for many applications in the process industry. 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. So far we have not seen any attempt in learning such networks. This paper has shown evolutionary programming to be a promising way of tackling this challenging problem.

Lam's Knowledge Guided Mutation operator made use of the description lengths of single links. We have extended this work to dynamic Bayesian networks where the number of possible networks can be very much greater. We have done this by proposing a new representation and by seeding the initial population with single links of low DL. We have also designed a new operator, *swap*, which maximises the recombination of these single links and exploits the smoothness of the DL graph of a single link with differing time lags. Our experiments appear to show that the performance of this operator is superior to the knowledge guided mutation operator when tested on synthetic time series data and real world oil refinery time series.

Further work will include:

1. Exploring how different evolutionary methods compare when various other Bayesian network metrics are used such as the Bayesian measure [Cooper 1991].
2. Learning a library of dynamic Bayesian networks and applying quickly-converging EPs on new refinery data in order to classify the current control structure of the refinery process.
3. Looking at ways to improve the accuracy of the networks such as looking at continuous Bayesian

networks or quickly learning a good discretization policy [Friedman 1996] through the use of an EP.

Acknowledgements

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, Andrew Ogden-Swift for his help in understanding the oil refinery data, and Stephen Swift for his general help and advice.

References

Baeck, T. Evolutionary Algorithms in Theory and Practice, Oxford University Press, 1996.

Cooper, G. Computational complexity of probabilistic inference using Bayesian belief networks (Research Note). Artificial Intelligence 42. pp 393-405, 1990.

Cooper, G., Herskovitz, E. A Bayesian Method for Constructing Bayesian Belief Networks from Databases, Proceedings of the 7th Conference on Uncertainty in AI, pp 86-94, 1991.

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

Fogel, D.B. Evolutionary Computation – Toward a New Philosophy of Machine Intelligence, IEEE Press, 1995.

Fogel, L.J., Owens, A.J., Walsh, M.J. Artificial Intelligence Through Simulated Evolution, John Wiley & sons, 1966.

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

Friedman, N., Murphy, K, Russell, S. Learning the Structure of Dynamic Probabilistic Networks, Proceedings of the 14th Conference on Uncertainty in AI, pp139-147, 1998.

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

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

Heckerman, D. A Tutorial on Learning with Bayesian Networks, Technical report, MSR-TR-95-06, November 1996.

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

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

Lam, W., Leung, K., Wong, M., Ngan, P. Discovering Probabilistic Knowledge from Databases using Evolutionary Computation and MDL Principal, Proceedings of the 3rd Annual Genetic Programming Conference, pp 786-794, 1998.

Larranaga, P., Poza, M., Yurramendi, Y., Murga, R., Kuijpers, C. Structure Learning of Bayesian Networks using GAs, IEEE Pattern Analysis and Machine Intelligence 18(9), pp 912-926, 1996.

Neapolitan, R. Probabilistic Reasoning in Expert Systems, Theory & Algorithms, Wiley, 1989.

Pearl, J. Probabilistic Reasoning in Intelligent Systems, Networks of Plausible Inference, Morgan Kaufmann, 1988.