An algorithm to learn the Structure of a Bayesian Network

Çiğdem Gündüz, Olcay Taner Yıldız, Ethem Alpaydın, Taner Bilgiç

Boğaziçi University
İstanbul 80815 Turkey

Abstract. We propose a new algorithm for learning the structure of a Bayesian network from given data. The algorithm creates the initial graph from zero and one independencies using a statistical test based on Kullback-Leibler mutual information. Then, by using graph operators such as removing or modifying the direction of an arc to increase the data likelihood, we arrive at the final graph. We perform several runs on the Alarm network and its subnetworks. We obtain promising results where our learned graphs show a good similarity to the original ones from which the data are sampled.

1 INTRODUCTION

A Bayesian network is a graphical model to encode the probabilistic relationships within a set of variables. A Bayesian network has the structure which is given by a directed acyclic graph (DAG) and the probability distributions [3].

Issues on the Bayesian networks are on learning the network [1] and finding efficient algorithm for inference on a given structure [9], [5], [6].

Learning consists of two main parts: The first problem is learning the structure of the model, which is finding the arcs (dependencies) between the nodes. The second problem is calculating the conditional and prior probabilities given the structure. In our paper, we discuss a new algorithm for learning the structure.

Structure learning algorithms can be classified into three: The first type of the algorithms are maximization problems where we want to find a belief network which maximizes a measure such as likelihood [4].

The second type of algorithms use independence criteria, where we want to find a network that represents as much as the original independencies in the data [9].

Third class of algorithms are hybrid of the former two, where the algorithms combine the scoring metrics and independence criteria [10].

Our algorithm can be classified into the third group because it finds the initial graph using the independencies in the data and modifies that graph by operators to maximize the likelihood of the data.

In Section 2, we discuss our algorithm in detail. Experiments are explained and results are given in Section 3. Section 4 contains the conclusion and the future work.

2 THE ALGORITHM

The probabilistic conditional independence relationships between the random variables indicate the absence of the arcs between the corresponding nodes on the Bayesian network. The variable X and Y are said to be conditionally independent given some variable Z, if and only if for all values of the variables X, Y and Z

$$P(X|Y,Z) = P(X|Z), \text{whenever} P(Y,Z) > 0 \tag{1}$$

where the cardinality of Z indicates the order of the conditional independency. If Z is empty, we have zero-independence, if Z is a singleton we have one-independence between X and Y, etc.

In our algorithm, we start by checking the zero and one independencies using a statistical test and we add edges between nodes where the dependency is high (with respect to a statistical test), starting from the highest, and adding until the graph is connected. After this step, we have a connected undirected graph. We then assign an initial ordering to the nodes and thus convert the graph to a connected directed graph. Then we apply two operators to modify the directed graph: One operator is *modify arc* which inverts the direction of an arc. The second operator is *remove arc*, which removes an arc from the graph. The criterion to be maximized in applying the operators is the data likelihood. At each step, we also make sure that the graph is connected and there is no cycle.

The pseudo-code of our algorithm is as follows:

```
1. for all variable tuples
     compute zero-independencies
     compute one-independencies
2. sort zero independencies in ascending order
3. specify the variable ordering using a heuristic
4. while graph is not connected
     select nodes with min zero-independencies
     if they are also not one-independent connect them in the given
     ordering
5. while there are changes
     select operator, arc number
     if the operator is modify arc
       compute the likelihood of data
       modify the direction of the arc
       compute the likelihood again
       if (likelihood increases OR likelihood decreases within a
       certain range) AND no cycle accept the modify operator
       else
         reject it
     if the operator is remove arc
       compute the likelihood of data
       remove the arc
```

```
        compute the likelihood again
        if likelihood increases and graph connected
          accept the remove operator
        else
          reject it
6. calculate the conditional and prior probabilities
```

2.1 OBTAINING THE UNDIRECTED GRAPH

The first step of our algorithm is to find the initial undirected graph from the training data. To find the initial graph, we must first determine where the edges are to be put. The edges are added according to the zero and one independencies of the data until the graph is connected.

For learning the zero and one independencies of the variable tuples from the data, we use the mutual information test which uses the Kullback-Leibler cross-entropy to measure the mutual information between the variables X and Y conditioned to the variables Z [7].

$$Inf(X,Y|Z) = \sum_{X,Y,Z} P(X,Y|Z) \log \frac{P(X,Y|Z)}{P(X|Z)P(Y|Z)} \tag{2}$$

The value $2 \times N \times Inf(X,Y|Z)$ approximates a χ^2 distribution with $|Z|(|X|-1)(|Y|-1)$ degrees of freedom, where N is the number of cases in the data set and $|A|$ represents the number of possible values for the variable A. This test is used to measure dependencies. If Z is empty, its cardinal value is zero, and $|Z|$ is taken to be one [2].

After we compute zero and one independencies between all possible variable tuples, we sort the zero independencies according to their confidence level. We begin to add the arcs between the nodes in ascending order until the graph becomes connected. The arcs are added only if the variables are also not one-independent.

De Campos and Huete (2000) also use the same order of independence tests to add *all* edges to the graph they are learning. They use a threshold for zero and one-independencies and they accept only those passing the threshold as dependent to start. Whereas we allow all zero-independencies and use a threshold for one-independency to start. But we stop whenever the graph is connected. It is expected that the dependencies that actually exist should rank higher in statistical tests. Therefore we allow all zero-independency tests to obtain a connected graph.

2.2 VARIABLE ORDERING

Even when we know which nodes are to be connected, still we do not know the directions of the arcs. The most important thing is not creating a cycle when assigning directions to the arcs. If there is a partial ordering between the variables, we can give the directions as follows.

Let us assume that there is an arc between a node X and a node Y. If variable X comes before variable Y in the ordering, the arc is directed from the node X to the node Y, otherwise the direction is reverse. This kind of direction assignment does not create a cycle.

Our ordering algorithm depends on the number of the values in the conditional probability tables of variables, because we want to keep the probability tables as small as we can. The pseudo-code of our ordering algorithm is as follows:

```
1. For each variable
     Assign all neighbor edges as incoming arcs.
     Compute the size of the conditional table according to the arcs.
     Mark variable as unselected.
2. i = number of arcs
3. While there are unselected nodes
     Select the node with the minimum table size.
     Give i to the selected node as ordering.
     Decrement i.
     Mark node as selected.
     Adjust the size of conditional tables of unselected nodes.
```

2.3 LEARNING STEPS

The backbone of the algorithm is the use of two operators. The initial graph is altered by these operators. The operators are removing an arc and modifying the direction of an arc. We decide to apply these operators by comparing the likelihood of the data before and after applying the operators. The likelihood of the data is the product of the joint probability distribution.

$$P(X_1, X_2, ..., X_n) = \prod_{foralldata} \prod_{i=1}^{n} P(X_i|pa(X_i)) \tag{3}$$

where $pa(X_i)$ denotes the parents of X_i.

We select the operator type randomly and the arc which this operator will be applied to is selected in order. Operators are always accepted when they improve the likelihood of the data. Modify operator is also accepted with a certain probability when the likelihood of the data decreases; this probability is decreased in time analogous to simulated annealing. Otherwise the operator and the corresponding arc is rejected and we try to find the better operator which improves the likelihood of the data. For applying the operators, the log-likelihoods (both before and after) are computed on the cross validation data set that was not used in the computation of dependencies step.

We do not want our graph to be unconnected, so the other criterion for the remove operator is to obtain always a connected graph. For the modify operator, we reject the operator if there is a cycle after applying it.

After we learn the structure, it is easy to fill the conditional tables because we assume no hidden variables or missing data. First we initialize each probability value in all conditional tables to zero. For each variable in the training set, we increment the joint probability value of the corresponding conditional table for that variable. If there is at least one nonzero value in a row of the conditional table, the values in that row are normalized. Otherwise each value in that row is found by dividing one to the number of states of that variable.

3 EXPERIMENTS AND RESULTS

For evaluating our algorithm, we perform several runs on the Alarm network and its subgraphs shown in Figures 1 and 2. The data from these networks with the known structure and the conditional probabilities are sampled to obtain training and cross validation sets. We perform several runs with the samples of different size.

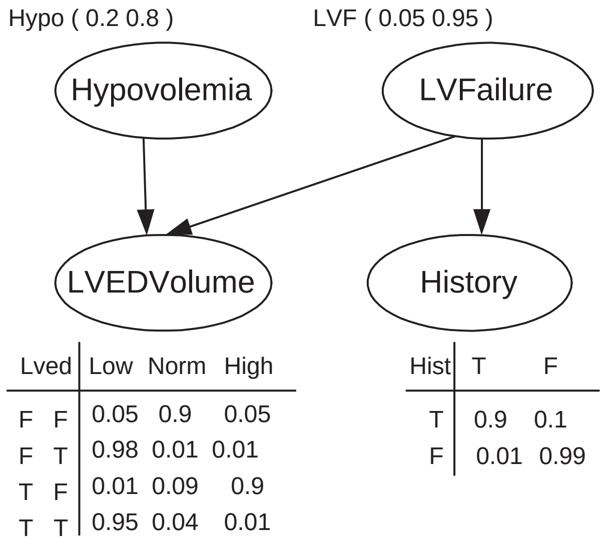
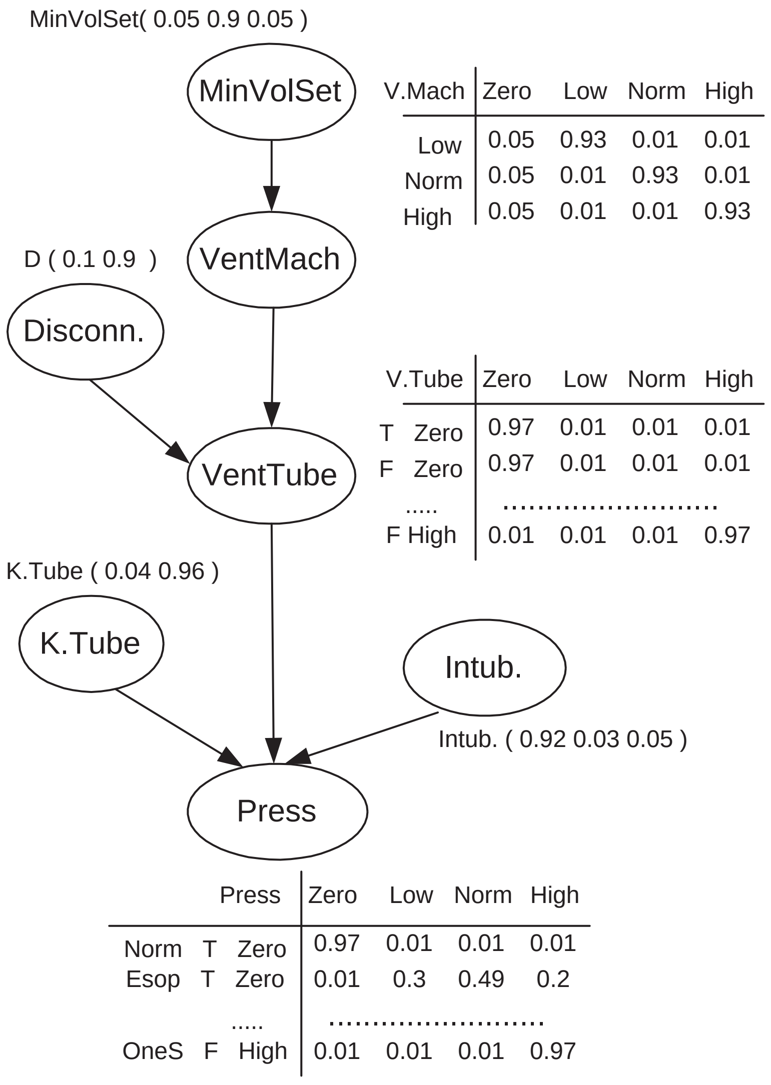


Figure 1. Subgraph of the Alarm network with four nodes

We use the training set to learn the initial graph, and we apply the operators by looking at the likelihood on the cross validation data.

From the different runs on the data sets, we can get different results. For the subgraphs, we observe that all original arcs are put in the initial graph. For the Alarm network we see that there can be missing arcs, however they are less. For both the Alarm network and its subgraphs, the directions of the arcs can be different from the original graphs. So putting the arcs until the graph becomes connected is enough most of the time. We do not have an operator to add an arc, because the arcs in the original graph have been already put in our initial graph most of the time. For the Alarm network, there is only one missing arc after the initial graph was formed. That is why we have only remove and modify arc operators.

While we apply the operators, we see that when the data size grows, the algorithm finds better graphs. For example, the remove arc operator is not so easy selected with small data. The final graphs that are obtained from large data sets have their arcs in



V.Mach	Zero	Low	Norm	High
Low	0.05	0.93	0.01	0.01
Norm	0.05	0.01	0.93	0.01
High	0.05	0.01	0.01	0.93

V.Tube	Zero	Low	Norm	High
T Zero	0.97	0.01	0.01	0.01
F Zero	0.97	0.01	0.01	0.01
.....	....			
F High	0.01	0.01	0.01	0.97

Press	Zero	Low	Norm	High
Norm T Zero	0.97	0.01	0.01	0.01
Esop T Zero	0.01	0.3	0.49	0.2
.....	....			
OneS F High	0.01	0.01	0.01	0.97

Figure 2. Subgraph of the Alarm network with seven nodes

the correct direction most of the time, but there can be extra arcs. When the data size grows, the number of these extra arcs also decreases.

The algorithm steps for the subgraph with four nodes are drawn in Figures 3– 5.

The algorithm steps for the subgraph with seven nodes are drawn in Figures 6– 8.

We have also run our algorithm on the original Alarm network. The final graph learned from a sample of 10,000 instances is shown in Figure 9. The original graph has 46 arcs. Our algorithm has only 3 missing arcs. 11 arcs are inverted and there are 23 extra arcs. It can be seen that our algorithm is able to extract the general structure from the data, with some extra arcs. These arcs can easily be seen and removed by an expert. Our experience with smaller networks has shown that spurious arcs are eliminated when the data sample gets larger; so another possibility is to run the algorithm on a larger sample. As given in [2], $d-$separation can also be used to remove unnecessary arcs.

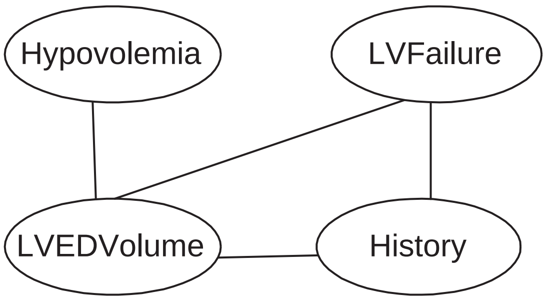


Figure 3. Initial graph after zero-one independencies

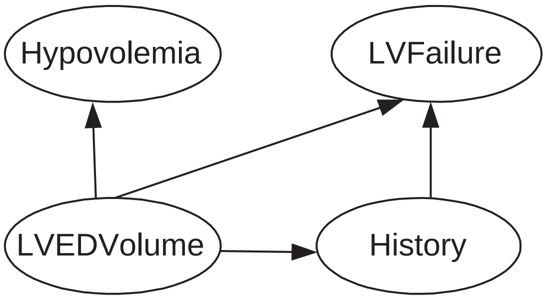


Figure 4. Initial graph before applying operators

4 CONCLUSION AND FUTURE WORK

In this paper we propose a novel algorithm to find the structure of a Bayesian network and learn its parameters from data. Simulation results show that our proposed algorithm gives good results as indicated by similar likelihoods with the original network, and similar structures with the right directions. Our algorithm heavily depends on data as all tests are based on statistical tests on the data.

In our algorithm, we assume the data does not contain any missing variables. If the data contains missing variables, they can be filled with the EM algorithm [8].

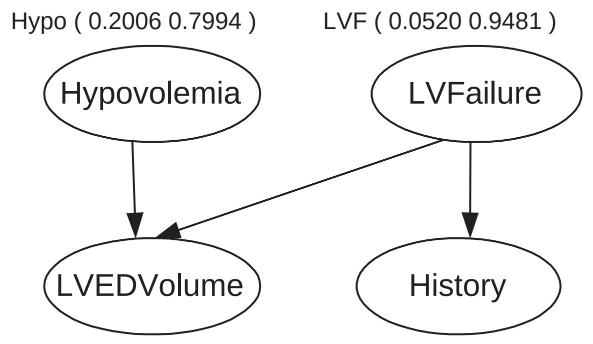
We can add further operators such as adding hidden nodes with appropriate arcs. In that case we can fill the values of the hidden nodes by EM.

To check the validity of our algorithm we can use several classification data sets and use the model we learned to make classifications.

In this paper we assumed that the variables are discrete. The algorithm can be extended to include continuous variables. This can be done in two ways, either by converting continuous variables into discrete or fitting continuous models to the continuous variables.

Acknowledgements

This work is supported by Boğaziçi University Research Funds Grant 00A101D.



Lved		Low	Norm	High
F	F	0.0524	0.9010	0.0486
F	T	0.9820	0.0097	0.0082
T	F	0.0106	0.0934	0.8961
T	T	0.9446	0.0428	0.0126

Hist	T	F
T	0.9051	0.0949
F	0.0109	0.9891

Figure 5. Final graph after applying operators and the calculated probabilities

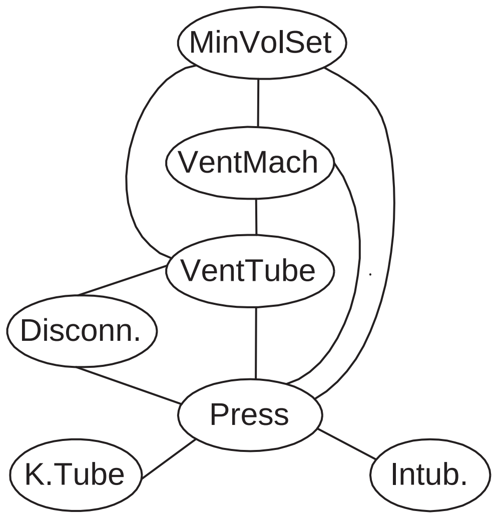


Figure 6. Initial graph after zero-one independencies

References

1. W. Buntine A Guide to the literature on learning probabilistic networks from data *IEEE Trans. Knowledge Data Engineering* Vol. 8, pp. 195-210, (1996).

2. L. M. de Campos and J. F. Huete A new Approach for learning belief networks using independence criteria *International Journal of Approximate Reasoning* Vol. 24, pp. 11-37, (2000).

3. D. Heckerman A Tutorial on Learning With Bayesian Networks Technical Report MSR-TR-95-06 Microsoft Research, (1995).

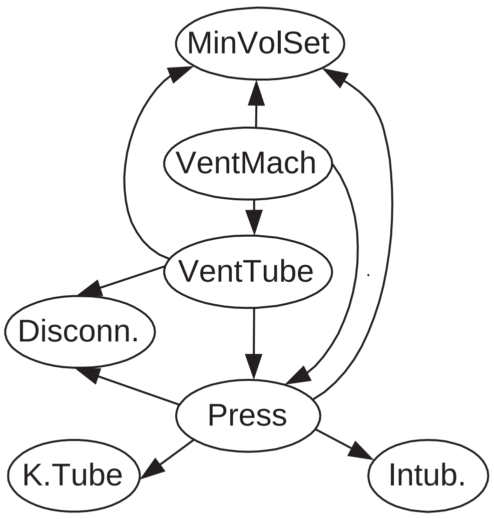
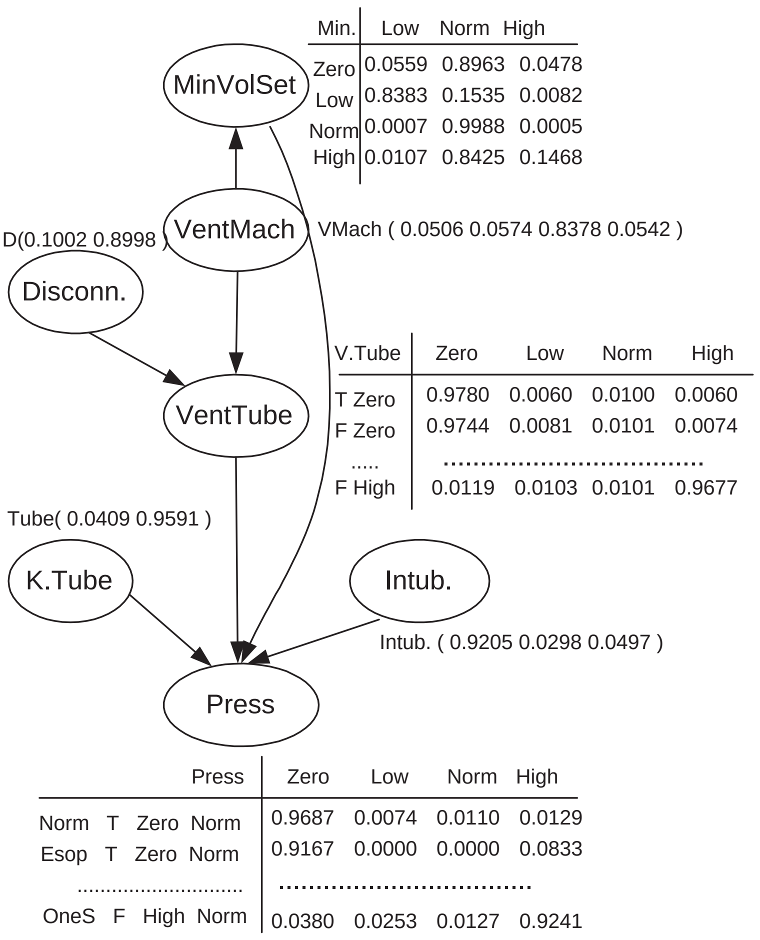


Figure 7. Initial graph before applying operators



Min.	Low	Norm	High
Zero	0.0559	0.8963	0.0478
Low	0.8383	0.1535	0.0082
Norm	0.0007	0.9988	0.0005
High	0.0107	0.8425	0.1468

V.Tube	Zero	Low	Norm	High
T Zero	0.9780	0.0060	0.0100	0.0060
F Zero	0.9744	0.0081	0.0101	0.0074
.....	....			
F High	0.0119	0.0103	0.0101	0.9677

Press	Zero	Low	Norm	High
Norm T Zero Norm	0.9687	0.0074	0.0110	0.0129
Esop T Zero Norm	0.9167	0.0000	0.0000	0.0833
....	....			
OneS F High Norm	0.0380	0.0253	0.0127	0.9241

Figure 8. Final graph after applying operators and the calculated probabilities

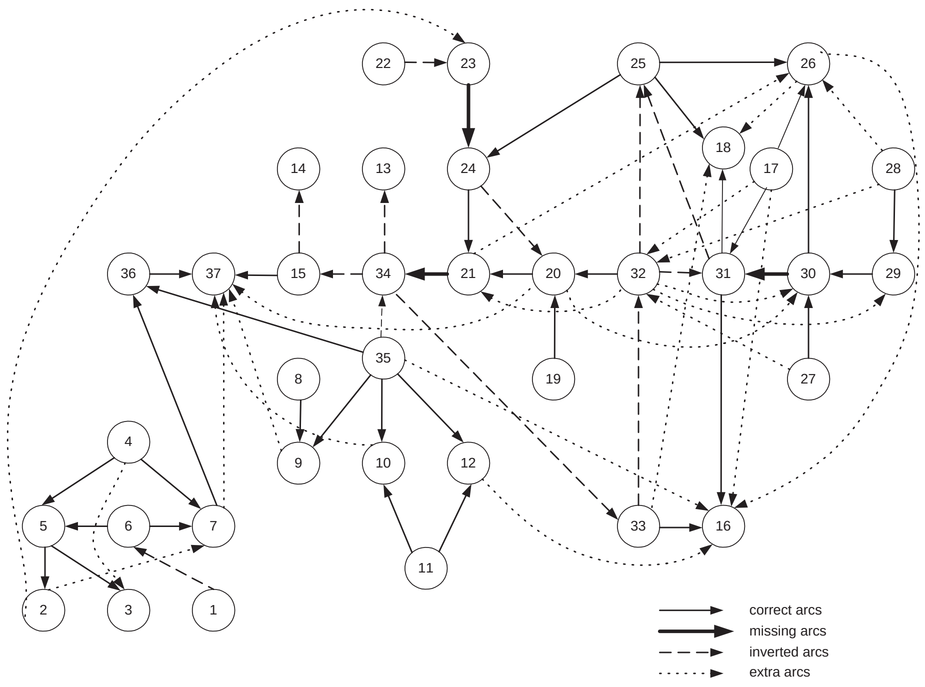


Figure 9. Final graph after applying operators to the original alarm network

4. D. Heckerman, D. Geiger and D. M. Chickering Learning bayesian networks: the combination of knowledge and statistical data *Machine Learning* Vol. 20, pp. 197-243, (1995).

5. C. Huang, A. Darwiche Inference in Belief Networks: A Procedural Guide *International Journal of Approximate Reasoning* Vol. 11, pp. 1-158, (1994).

6. F. Jensen An Introduction to Bayesian Networks Springer, (1996).

7. S. Kullback Information Theory and Statistics Dover, New York, (1968).

8. S. L. Lauritzen The EM algorithm for graphical association models with missing data *Computational Statistics and Data Analysis* Vol. 19, pp. 191-201, (1995).

9. J. Pearl Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference Morgan Kaufmann, San Mateo, (1988).

10. P. Spirtes, T. Richarson, C. Meek Learning Bayesian Networks with discrete variables from data *Proceedings of the First International Conference on Knowledge Discovery and Data Mining* pp. 294-299, (1995).