Stochastic Techniques in Influence Diagrams for Learning Bayesian Network Structure

Michal Matuszak[1] and Jacek Miękisz[2]

[1] Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Chopina 12/18, 87–100 Torun, Poland
gruby@mat.umk.pl
[2] Institute of Applied Mathematics and Mechanics, University of Warsaw, Banacha 2, 02–097 Warsaw, Poland
miekisz@mimuw.edu.pl

Abstract. The problem of learning Bayesian network structure is well known to be NP–hard. It is therefore very important to develop efficient approximation techniques. We introduce an algorithm that within the framework of influence diagrams translates the structure learning problem into the strategy optimisation problem, for which we apply the Chen's self–annealing stochastic optimisation algorithm. The effectiveness of our method has been tested on computer–generated examples.

Keywords: Bayesian Networks, Structure Learning, Chen Adaptive Optimisation, Influence Diagrams

1 Introduction

Bayesian networks represent probabilistic relationships among given variables. They are built on directed acyclic graphs (DAG) in which nodes represent random variables (ovals in our figures below) and direct edges between nodes represent the probabilistic dependencies between them. Conditional probabilities for variables are stored in conditional probability potentials (or tables) attached to dependent nodes.

There are two basic learning problems in Bayesian networks: learning the structure of a graph and learning the conditional probability potentials. It is fairly easy to learn parameters of a given DAG (see [6,8,9,12]). One approach is to compute frequencies that are optimal with respect to the maximum likelihood estimation (MLE). Here we focus on the task of learning the structure of a DAG from a given dataset. It has many important applications in various fields like classification and variable selection, and bioinformatics, where it is used for locating gene regulatory pathways (check [9] for more applications).

Learning Bayesian network structure is NP–hard even for networks with two parents [4]. Chow and Liu [5] showed that trees can be learned in a polynomial time but it has been shown in [7] that even learning 2–polytrees is an NP–hard

problem. A polytree is a DAG with the property that if the directions on edges are ignored, it results in an undirected graph with no cycles.

There are many learning algorithms (see [6,8,9,12]) for building structure of Bayesian networks. Generally, they can be divided into three main groups: `constraint based` algorithms, `search and score` techniques, and `hybrid` methods which combine the first two methods. The constraint based algorithms perform a study of the dependence and independence relationships among variables of the Bayesian network. They are performed by conditional independence tests. For large enough datasets, χ^2 or G^2 tests can be performed and for the smaller ones exact tests could be done. Main problems of those algorithms are: the time complexity of the independence tests, unreliable results of the independence tests, and also the fact that the most widely used algorithms require an existence of a faithful graph [9].

The search and score techniques attempt to find a graph structure that maximizes the value of a given scoring function. A brief description of these techniques is provided in the next section.

An influence diagram is an extension of a Bayesian network. They both are built on DAG's and consist of chance nodes, while influence diagrams also have decision and utility nodes. They not only provide tools for a probabilistic inference but also provide a language for sequential decision making problems, where there is a fixed order among the decisions.

In [3], a simple stochastic optimisation algorithm for the traveling salesman problems (TSP) has been proposed. Then in [11], Chen's ideas have been substantially extended in order to construct an algorithm for solving general influence diagrams. It shows a strong performance in optimal transport problems [10]. In this paper, we translate the structure learning problem into the one involving influence diagrams and we show that it can be solved with an extended version of the algorithm presented in [11].

2 Learning Bayesian networks structure

Formally, a Bayesian network is a pair (G, θ), where $G = (C, E)$ is a directed acyclic graph (DAG), $C = \{X_1, \ldots, X_n\}$ consists of n random variables, E represents direct dependencies between variables, and θ represents a set of parameters for each variable in C, which defines their conditional probability distributions. Each random variable X_i has values in a finite domain K_i.

The problem of learning a Bayesian network structure is given as follows: For a set of random variables $C = \{X_1, \ldots, X_n\}$ and a database of m cases $M = \{M_1, \ldots, M_m\}$, where each case contains observations of all variables in C, i.e. $M_i = (x_1, \ldots, x_n)^T$ is a vector of instances of variables $X_1, \ldots, X_n$, find a DAG (that is a set of directed edges) which best matches M.

As stated before, we focus our attention on `search and score` techniques. A search space of possible DAG's grows super–exponentially [14], so testing all possible DAG patterns is computationally unfeasible. Scoring functions can be

divided into two main classes: `Bayesian scoring` functions such as K2, the mutual information test (MIT), the Bayesian Dirichlet test and its variants (BD, BDe, BDeu), and `information-theoretic` scoring functions such as the log likelihood (LL), the Bayesian information criterion (BIC), and the Akaike information criterion (AIC).

Here we will use a modification of the Cooper–Herskovits likelihood (belonging to the Bayesian scoring class) [6] for a DAG G and a dataset M with $P(G)$ as a prior probability of G. It has the following form,

$$P(G, M) = P(G) \times \prod_{k=1}^{n} \prod_{j=1}^{|\phi_k|} \frac{(s_k - 1)!}{(s_{kj} + s_k - 1)!} \prod_{l=1}^{s_k} \alpha_{kjl}! \tag{1}$$

where s_k is a number of states of the variable X_k, ϕ_k is a variable describing joint configurations of variables in $\pi(X_k)$ ($\pi(X_k)$ denotes the set of parents of X_k), $|\phi_k|$ is a number of states of ϕ_k, and α_{kjl} is a number of cases in M in which X_k is at the l–th state and ϕ_k is at the j–th state. Also $s_{kj} = \sum_{l=1}^{s_k} \alpha_{kjl}$.

The evaluation of a broad spectrum of scoring functions can be found in [2]. It is stated there that there are only small differences between various scoring functions and all of them behave in a similar way (only the BIC score was clearly the worst). Thus, we use the K2 metric [6] which is a slight modification of the Cooper–Herskovits likelihood from Eq. 1. Its objective is to find the most probable network structure, with a given data set, which maximizes the posterior probability distribution. It assumes a uniform prior $P(G)$ and instead of $P(G, M)$ uses $log(P(G, M))$.

3 The algorithm

Now we give a formal description of our algorithm. Let us assume that a set of chance variables $C = \{X_1, \ldots, X_n\}$ and a database of cases $M = \{M_1, \ldots, M_m\}$ are given. Both objects are fixed during the execution of the algorithm. In addition, to each chance variable $X_i \in C$ a decision node D_i is attached. Decision nodes play crucial role in the algorithm. During the optimization procedure, they unfold a structure, that is a set of directed edges E of the Bayesian network. Our algorithm is based on an extension of ideas from [3,11].

If nodes ordering is not given, then each decision node D_i has n states. First $n - 1$ states describe available connections with chance variables $C \setminus X_i$ and the $n-$th state is used during the optimization process and is applied to disable connections of further children. The proposed algorithm does not require a node ordering, however, it may benefit from a predetermined ordering as the search space will be reduced.

For each decision node D_i, a *randomised policy* τ_i is attached. It assigns probabilities to all possible decisions that may be taken, with $\tau_i^{X_j}$ standing for the probability of adding a direct edge from X_i to X_j. In the course of the optimisation process, these randomised policies evolve and include an (sub)optimal

structure of the network. The initial choice of τ_i can be either *uniform* or *heuristic*. In the *uniform* choice, all decisions are equiprobable and in the *heuristic* case, some additional knowledge is provided allowing us to make a good first guess about the optimal structure. For example if we know that an edge is more probable than others starting from the same node, then we can increase it's probability and thus making the edge appear with a higher probability in the optimization procedure. If we have a knowledge that some edges exist in the network, then we can add them as *permanent edges* and they will always be included in the network structure.

[**Permanent edge condition**] During the iterative procedure, the *randomised policies* should converge to Dirac deltas which results in an almost deterministic selection of edges. It means for the node $X_i \in C$, that if

$$1 - \tau_i^{X_j} < \epsilon \tag{2}$$

for some node $X_j \neq X_{None}$ and a small fixed ϵ, a direct edge $X_i \to X_j$ will be added to E. However, if $X_j = X_{None}$, then X_i is permanently removed from active nodes, resulting in no further addition of children. This does not however restricts its possibility of being a child.
It is possible that two (or more) decisions are equiprobable and then τ_i is almost a uniform distribution over them, while the probabilities of other decisions are near 0. Therefore, if Eq. 2 is not satisfied after a fixed number of steps Q, we should randomly choose a decision (that is a vertex to connect) according to the probability distribution τ_i and if decision X_{None} is drawn, then we deactivate X_i, and in other cases we add a direct edge $X_i \to X_j$ to E.

Each decision node has also an *active* field with states $\{enabled, disabled\}$ which describe whether the node is subject to the optimization procedure. At the initialization of the algorithm, all nodes are *active*. When node's optimization is over (see *permanent edge condition*), then it becomes passive and it can not be reactivated.
The algorithm works as follows.

1. Set the iteration counter $j = 0$.
2. Attach and initialize decision nodes as described above.
3. Generate an instance of the network:
   (a) If predefined edges are given, include them in the network and follow the actions from *permanent edge condition*.
   (b) Select randomly (with a uniform distribution without replacement) an active node D_i.
   (c) According to the distribution τ_i draw a new vertex X_h and if $X_h \neq X_{None}$ and a direct edge from X_i to X_h preserve acyclicity of the graph, then add that edge.
   (d) Go back to Step 3b until each active node is chosen.

4. Select randomly (with a uniform distribution) an active node D_i and denote its state as h_0. It is an index that can be used to select the h_0–th chance node (X_{h_0}) or h_0–th decision node (D_{h_0}).
   (a) If h_0 corresponds to an existing edge ($X_i \to X_{h_0}$), then remove the edge.
   (b) According to the probability distribution τ_i draw a new vertex X_{h_1} and if $X_{h_1} \neq X_{None}$, then add a direct edge from X_i to X_{h_1}. In other words, during the search space step we can add or remove an edge.
   (c) Accept the modification if the following conditions are met:
      – a new vertex X_{h_1} is different from the current vertex X_{h_0}.
      – a graph is acyclic.
      – the weakly connected property is preserved i.e. the skeleton (undirected graph obtained from replacing directed edges with undirected ones) is connected. The property can be verified with using either depth-first or breadth-first search algorithm.

   else go to Step 4.
5. Evaluate the utility function $U^{(j)}$ using, for example, the K2 metric.
6. Check the *permanent edge condition* and if $X_i \to X_{h_1}$ has been added to E, remove from the support of τ_i the decision to link to the node X_{h_1} and from the support of τ_{h_i} the decision to link to the node X_i . Reinitialize the values of τ_i with the uniform distribution (or apply a predefined knowledge to the distribution). For τ_{h_1}, normalize the weights (by dividing them by their sum) to achieve a probability distribution.
7. IF $j \bmod B \neq 0$ set
$$\Delta := U^{(j)} - U^{(j-1)} \tag{3}$$
then update the randomised policy like in the Chen's algorithm,
$$\tau_i^{X_{h_1}} = \exp(\beta\Delta)\tau_i^{X_{h_1}}, \tag{4}$$
and renormalise τ_i so that it remains a probabilistic distribution. The important parameter of the algorithm is constant β which describes the rate of learning (larger β speeds up learning but decreases the optimization's stability). β represents the noise level in the algorithm, it corresponds to the inverse of the temperature in physical systems.
8. Set $j = j + 1$ and if there still exists an active node then:
   – if $j \bmod B = 0$ return to Step 3. For small enough β the initial order of D_i has only a small impact on the algorithm. However, in rare situations even for small β impact of the first selection of the nodes could cause divergence of the algorithm. To resolve that issue and to allow for larger values of β (which speed up the optimization) we implement *restarting after fixed number of steps* i.e a new order of nodes is generated after B steps.
   – else return to Step 4

In Fig. 1, we present an application of our algorithm to a network with three chance nodes: X_1, X_2, and X_3. After the initialization phase the decision nodes

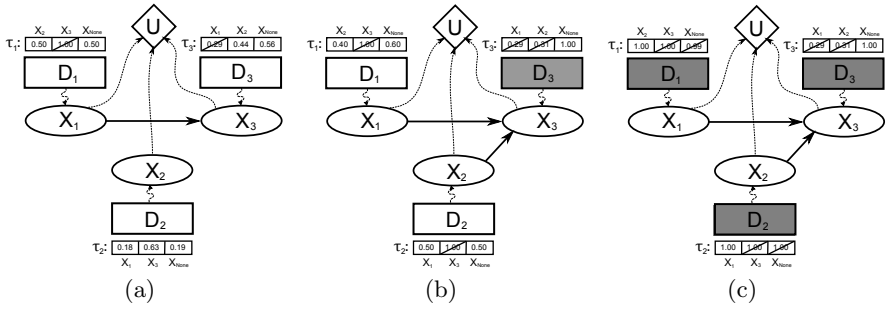

Fig. 1. An example of the algorithm executed on a network with three chance nodes.

and randomised policies have been attached. The utility node U, which computes the score function, has been added and connected with chance nodes. In our example, no prior knowledge is available, thus the randomised policies are represented by uniform distributions. All nodes are active and the algorithm can proceed. An instance of the network is generated and during the iterative procedure the search space of possible DAG's is explored, and randomised policies τ_i are modified with the use of the Chen's formula [3]. Fig. 1(a) presents the network state after $\tau_1^{X_3}$ converged to 1. A permanent direct edge $X_1 \rightarrow X_3$ is added, the policies ($\tau_1^{X_3}$ and $\tau_3^{X_1}$) associated with it are deleted, τ_1 is initialized with the uniform distribution and weights in τ_3 are renormalized.

In Fig. 1(b), $\tau_2^{X_3}$ converges to 1. A permanent direct edge $X_2 \rightarrow X_3$ is added and the policies $\tau_2^{X_3}$, and $\tau_3^{X_2}$ are deleted. The only available policy in τ_3 is $\tau_3^{X_{None}}$ so no further children can be connected to X_3 and the decision node D_3 can be *deactivated* (excluded from the optimisation procedure). Weights in τ_1 are initialized with the uniform distribution and we return to the iterative procedure. Fig. 1(c) presents the network status after $\tau_2^{X_{None}}$ converged to 1, so no edge is added and D_2 is *deactivated*. The only possible edge that still can be added is $X_1 \rightarrow X_2$, but $\tau_1^{X_{None}}$ also converged to 1, and D_1 is *deactivated*. All decision nodes are inactive, thus the algorithm stops and returns a set of directed edges E.

4 Numerical examples

The programme has been implemented in the language `C++`, with the implementation aimed so far mainly at algorithm evaluation purposes, it can be described as careful but not fully performance–optimised.

We have selected two networks for numerical experiments: a *simple Bayesian network* with 7 nodes and 7 edges (Fig. 2(a)) and *ALARM network* [1] which contains 37 nodes and 46 edges (Fig. 2(b)). Each network has been used to generate a database, which contains 100000 instances.

For the *simple Bayesian network* with $\beta = 0.001$, $\epsilon = 0.01$, $B = 100$ and the number of iterations limited to $Q = 500$, a final network differs from the optimal

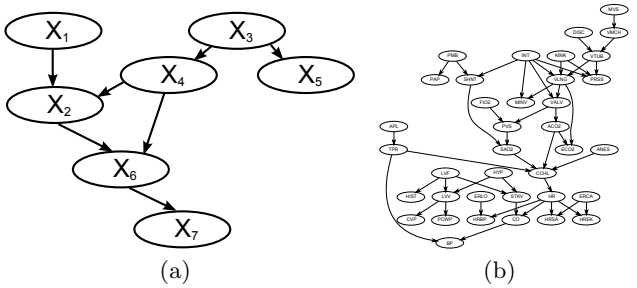

Fig. 2. (a) Simple Bayesian network; (b) ALARM network.

one in K2 metric by 0.11% (for the Chow–Liu tree it is 0.73%, and for classical K2 algorithm [6] it is 1%), and for the *ALARM network* with $\beta = 9 \times 10^{-6}$, $\epsilon = 0.01$, $B = 250$ and iterations limited to $Q = 15000$, the difference in K2 metric is 20% (16% for the Chow–Liu tree, and for classical K2 algorithm it is 33%).

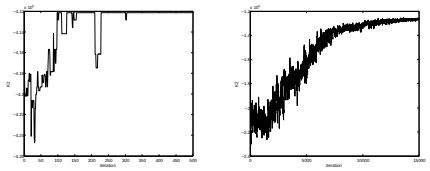

Fig. 3. Convergence of the algorithm for (a) Simple Bayesian network and (b) ALARM network.

	Simple net	ALARM net
K2	0.195	99
Chow–Liu	0.1	12
Our algorithm	0.9	917

Table 1. Computational times (in seconds).

In Fig. 3, we show the convergence of our algorithm, that is the maximization of the K2 metric. In Table 1, we compare computational times of our and other algorithms.

5 Conclusions

A new stochastic algorithm for finding Bayesian network structure has been presented. Our method is based on an innovative application of Bayesian influence diagrams for structure optimization. The main advantage of the introduced method is the use of an algorithm that can determine an optimal decision strategy for a different problem. Numerical results indicate the correctness of the presented algorithm. Although computational times of our algorithm are not optimal (see Table 1) our results are competitive as compared to classical ones.

Acknowledgements

This research has been supported by the National Science Centre grant 2011/01/N/ST6/00573 (2011-2014). The authors gratefully acknowledge the access to the PL–Grid[3] infrastructure, that is co–funded by the European Regional Development Fund as a part of the Innovative Economy program. The work of M. Matuszak has also been supported by the European Social Fund as a part of the Sub–measure 4.1.1 (National PhD Programme in Mathematical Sciences).

References

1. Beinlich, I., Suermondt, H. J., Chavez, R. M., and Cooper, G. F. The ALARM monitoring system: a case study with two probabilistic inference techniques for belief networks, In Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine, pp. 247–256 (1989).
2. de Campos, L. M. A scoring function for learning Bayesian networks based on mutual information and conditional independence tests, Journal of Machine Learning Research 7, pp. 2149–2187 (2006).
3. Chen, K. Simple learning algorithm for the traveling salesman problem, *Phys. Rev. E* **55**, 7809-7812 (1997).
4. Chickering, D. M. Learning Bayesian networks is NP–complete, Learning from Data: Artificial Intelligence and Statistics V, pp. 121–130. Springer–Verlag (1996).
5. Chow, C. K., Liu, C. N. Approximating discrete probability distributions with dependence trees. IEEE Trans. Info. Theory, 14(3), pp. 462–467, (1968).
6. Cooper, G.F., Herskovits, E. A Bayesian method for the induction of probabilistic networks, Data Machine Learning vol. 9, pp. 309 – 347 (1992).
7. Dasgupta, S. Learning polytrees, Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, pages 131–141, Morgan Kaufmann (1999).
8. Jensen, F.V., Nielsen, T.D. Bayesian Networks and Decision Graphs, 2nd Ed., *Springer*, (2007).
9. Koski, T., Noble, J. Bayesian Networks: An Introduction, John Wiley & Sons, Ltd (2009).
10. Matuszak, M., Miękisz, J., Schreiber, T. Solving Ramified Optimal Transport Problem in the Bayesian Influence Diagram Framework, ICAISC 2012, LNAI, (2012).
11. Matuszak, M., Schreiber, T. A new stochastic algorithm for strategy optimisation in Bayesian influence diagrams, ICAISC 2010, Part II, LNAI 6114, pp. 574-581 (2010).
12. Neapolitan, R. E. Learning Bayesian Networks, *Prentice Hall Series in Artificial Intelligence*, *Pearson Prentice Hall*, (2004).
13. Peretto, P. An Introduction to the Modeling of Neural Networks, *Collection Aléa–Saclay*, *Cambridge University Press* (1992).
14. Robinson, R.W. Counting unlabelled acyclic digraphs, *Springer Lecture Notes in Mathematics: Combinatorial Mathematics V*, *C.H.C. Little (ed.)*, pp. 28 – 43 (1977).

[3] `http://www.plgrid.pl`