The Hugin Tool for Learning Bayesian Networks

Anders L. Madsen, Michael Lang, Uffe B. Kjærulff, and Frank Jensen

Hugin Expert A/S
Niels Jernes Vej 10
DK-9220 Aalborg Ø
Denmark
{Anders.L.Madsen,Michael.Lang,Uffe.Kjaerulff,Frank.Jensen}@hugin.com

Abstract. In this paper, we describe the Hugin Tool as an efficient tool for knowledge discovery through construction of Bayesian networks by fusion of data and domain expert knowledge. The Hugin Tool supports structural learning, parameter estimation, and adaptation of parameters in Bayesian networks. The performance of the Hugin Tool is illustrated using real-world Bayesian networks, commonly used examples from the literature, and randomly generated Bayesian networks.

1 Introduction

Probabilistic graphical models such as Bayesian networks [9,3] are efficient models for (automated) reasoning under uncertainty. A Bayesian network can be used as an efficient tool for knowledge representation and inference. Unfortunately, the construction of a Bayesian network can be a quite labor intensive task to perform. For this reason, automated construction of Bayesian networks have in recent years received a lot of attention. This attention has focused on the automated construction of models from a combination of data and domain expert knowledge. In this paper, we consider the model construction task as a task of fusing observational data and domain expert knowledge. Through automated construction, Bayesian networks can be used as efficient tools for knowledge discovery and data mining [5].

The Hugin Tool [1,6] is a general purpose tool for probabilistic graphical models such as Bayesian networks and influence diagrams. In this paper, we describe the knowledge discovery functionality of the Hugin Tool related to (automated) construction of Bayesian networks through learning. That is, we describe the capabilities of the Hugin Tool for learning the structure and parameters of a Bayesian network. In [6] a recent survey of the general functionality of the Hugin Tool is given. The present paper extends and details the description of the learning functionality of the Hugin Tool given in [6].

2 Preliminaries and Notation

A Bayesian network $\mathcal{N} = (G = (V, E), \mathcal{P})$ consists of an acyclic, directed graph (DAG) G and a set of probability distributions $\mathcal{P}$. Each node $X \in V$ represents a

T.D. Nielsen and N.L. Zhang (Eds.): ECSQARU 2003, LNAI 2711, pp. 594–605, 2003.

unique random variable. (We use the terms "node" and "variable" interchangeably and consider only discrete variables.) For each variable $X \in V$ there is a conditional probability distribution $P(X \mid \mathrm{pa}(X)) \in \mathcal{P}$.

A Bayesian network $\mathcal{N} = (G, \mathcal{P})$ is an efficient representation of a joint probability distribution $P(V)$ over V when $G = (V, E)$ is not dense. $P(V)$ factorizes according to the structure of G as:

$$P(V) = \prod_{X \in V} P(X \mid \mathrm{pa}(X)).$$

We denote variables by uppercase letters $X, Y, \ldots$, sets of variables by uppercase letters $S, S_{XY}, \ldots$, and states of variables by lowercase letters $x, y, \ldots$. Through the parameters, it is possible to specify unconstrained probabilistic dependence relations between a node and its parents.

The graph G of $\mathcal{N}$ induces a set of (conditional) dependence and independence relations (CIDRs) $\mathcal{M}_G$, which can be read off G using the d-separation criteria [4]. The relation $X \perp_P Y \mid S$ states conditional independence between X and Y given S under the probability distribution P whereas $X \perp_G Y \mid S$ states conditional independence between X and Y given S in the DAG G (i.e. d-separation). When no confusion is possible, subscripts are omitted. The faithfulness assumption [13] (a.k.a. stability [10]) says that the distribution P over V induced by (G, Θ) satisfies no independence relations beyond those implied by the structure of G.

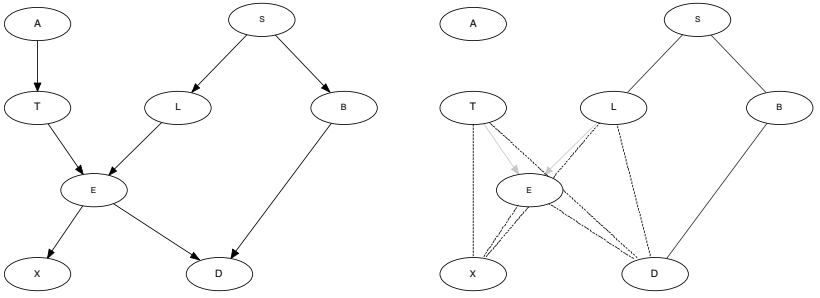
A DAG represents a set of CIDRs and two DAGs may represent the same set of CIDRs. Two DAGs representing the same set of CIDRs are equivalent. A DAG is an acyclic, directed graph whereas a PDAG is an acyclic, partially directed graph, i.e. an acyclic graph with some edges undirected (a.k.a. a pattern [10]). A PDAG can be used to represent the equivalence class of a DAG. The equivalence class of a DAG G is the set of DAGs with the same set of d-separation relations as G. Two DAGs G_1 and G_2 are equivalent if they have the same skeleton and the same set of colliders (i.e. $X \to Y \leftarrow Z$-structures), see e.g. [10].

Example 2.1 [Chest Clinic]
Dyspnoea(D) may be due to tuberculosis(T), lung cancer(L), or bronchitis(B), or none of them, or more than one of them. A recent visit to Asia(A) increases the chances of tuberculosis, while smoking(S) is known to be a risk factor for both lung cancer and bronchitis. The result of a single chest X-ray(X) does not discriminate between lung cancer and tuberculosis, as neither does the presence or absence of dyspnoea, see e.g.[3].

The qualitative knowledge of this diagnostic problem can be captured by the DAG shown in Fig. 1(a) with a mediating variable E representing the disjunction of tuberculosis and lung cancer.

3 Learning a Bayesian Network

In the remainder of this paper, we will assume that P_0 is a DAG faithful probability distribution with underlying DAG G_0. We consider learning a Bayesian



(a) Specification (b) Result of NPC

Fig. 1. The Chest Clinic example

network as the task of identifying a DAG structure G and a set of corresponding parameters Θ from a sample of data cases $D = \{c_1, \ldots, c_N\}$ drawn at random from P_0 and possibly some domain expert background knowledge.

3.1 Structural Learning

Structural learning is supported through a constraint-based approach [16,15, 13]. In the constraint-based approach, the graph G of a Bayesian network $\mathcal{N}$ is considered as an encoding of a set of CIDRs $\mathcal{M}$. Structural learning is then the task of identifying a DAG structure from a set of CIDRs derived from the data by statistical tests.

Two algorithms for structural learning are supported. The PC algorithm [12, 13] (which is similar to the IC algorithm [15,10]) and its extension, the NPC algorithm [14]. The main steps of the PC algorithm are:

1. Test for (conditional) independence between each pair of variables.
2. Identify the skeleton of the graph induced by the derived CIDRs.
3. Identify colliders.
4. Identify derived directions.

The PC algorithm produces a PDAG. In step 1, the hypothesis is that X and Y are independent given S_{XY}. This hypothesis is tested by statistical tests using conditioning sets S_{XY} of size $0, 1, 2, 3$. If $X \perp Y \mid S_{XY}$ is found to be satisfied with some significance level α, the search for independence between X and Y is terminated.

Various improvements of the straightforward incremental testing scheme have been implemented. These improvements are related to maintaining an undirected graph describing the current set of neighbors of each node and only performing independence tests conditional on subsets of the neighbors of X and Y. The order in which we try out the possible conditioning sets of a fixed size is according to how likely they are to cause independence for the edge under consideration. We

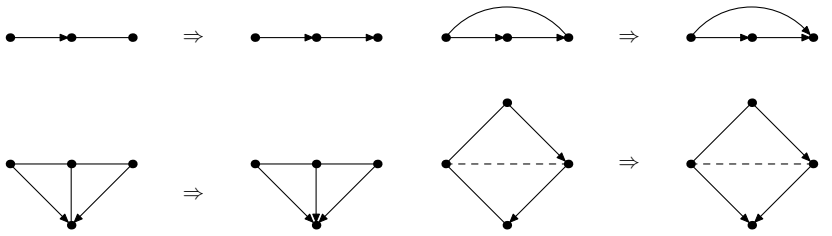


Fig. 2. Four rules for orientation of edges

use the heuristic rule that the variables of the conditioning set should be strongly correlated with both endpoints of the edge being tested. The neighbor graph is updated after each independence test accepting the hypothesis.

Due to the nature of the testing scheme, the conditioning set S_{XY} for an identified independence relation $X \perp Y \mid S_{XY}$ is minimal in the sense that no proper subset of S_{XY} and no set of cardinality less than the cardinality of S_{XY} produce independence. An undirected edge is added between each pair of variables X, Y whenever no conditional independence relation has been found between X and Y. This produces the skeleton of the graph.

Once the skeleton has been identified, colliders are identified. If X and Y are neighbors, Z and Y are neighbors, X and Z are not neighbors, and $Y \notin S_{XZ}$ for any S_{XZ} satisfying $X \perp Z \,|\, S_{XZ}$, then a collider is created at Y.

Starting with any PDAG G, a maximally directed PDAG can be obtained following four necessary [15] and sufficient [8] rules, see Fig. 2. That is, by repeated application of these four rules all edges common to the equivalence class of G are identified. The fourth rule is unnecessary, if the orientation of the initial PDAG is limited to colliders (i.e. no background knowledge). The four rules are necessary and sufficient for achieving maximal orientation (up to equivalence) of the PDAG returned by the PC algorithm. The first rule follows from the fact that no collider was identified, the remaining rules ensure that no directed cycle is created.

Correctness of the PC algorithm has been proved under the assumption of infinite data sets. In real-life, data sets are finite. When dealing with finite data sets, the faithfulness assumption is often violated. Hence, when the derived set of CIDRs is induced by statistical tests on finite data sets, we cannot in general expect that there exists a DAG (or PDAG) which represents all CIDRs. Often too many conditional independence relations are derived due to the limited data set. This suggests to represent all conditional dependence relations, but not all conditional independence relations in the induced DAGs. Applying the principle of Occam's Razor, we will choose the simplest model among equally good models.

As mentioned above, the NPC algorithm is an extension of the PC algorithm. The new feature of the NPC learning algorithm is the introduction of the notion of a *Necessary Path Condition* [14]. Informally, the necessary path condition says that in order for two variables X and Y to be independent (in a DAG faithful

data set) conditional on a set S and no subset $S' \subset S$, there must exist a path between X and every $Z \in S$ (not crossing Y) and between Y and every $Z \in S$ (not crossing X). Otherwise, the inclusion of each Z in S is unexplained. Thus, in order for an independence relation to be valid, a number of edges are required to be present in the graph.

An edge (X, Y) is an uncertain edge, if the absence of (X, Y) depends on the presence of an edge (X', Y'), and vice versa. A maximal set of interdependent uncertain edges is an ambiguous region. An uncertain edge indicates inconsistency in the set of independence relations derived by the statistical tests.

In order to increase reliability and stability of the NPC algorithm, the iteration step for a fixed size of the conditioning set is completed even if an independence statement is found. Thus, multiple independence relations may be found for a pair of variables. If one of these independence relations satisfy the necessary path condition, then it is accepted.

Prior to the testing phase, background knowledge in the form of constraints on the structure of the DAG can be specified. It is possible to specify the presence and absence of edges, the orientation of edges, and a combination. At the moment, user specified constraints are not tested. In practice, this has produced some unwanted behavior of the edge orientation algorithm.

Example 3.1 [Structural learning in Chest Clinic]
Figure 1(b) shows the PDAG generated by the NPC algorithm applied on a random sample of $10,000$ *cases drawn from the Chest Clinic network with a significance level of* $\alpha = 0.05$.

The sets of edges $\{(T, X), (E, X), (L, X)\}$ *and* $\{(T, D), (E, D), (L, D)\}$ *are the two ambiguous regions of Fig. 1(b). The two ambiguous regions are due to the deterministic relation between* E *and* L, T *(i.e.* $E = L \vee T$*). This produces, for instance,* $\{(X \perp E \mid T, L), (X \perp T \mid E), (X \perp L \mid E)\}$*, which is impossible according to the necessary path condition. The simplest resolution is to include the edge* (E, X)*. Notice that some certain edges are directed and some are undirected.*

3.2 Parameter Estimation

The task of parameter estimation is to estimate the values of the parameters Θ corresponding to a given DAG structure G. Parameter estimation is supported through the EM algorithm [7]. The EM algorithm is well-suited for calculating maximum likelihood and maximum a posteriori estimates in the case of missing data.

Let $\mathcal{N} = (G, \mathcal{P})$ be a Bayesian network with parameters Θ such that $\theta_{ijk} = P(X_i = k \mid \mathrm{pa}(X_i) = j)$ for each i, j, k. Following [7] the EM algorithm is based on computing the expected value of the log-likelihood function:

$$Q(\Theta^* \mid \Theta) = \mathbb{E}_\Theta\{\log P(X \mid \Theta^*) \mid D\},$$

where P is the density function for X, and D is the observed data $D = g(X)$. Given an initial value of the parameters Θ, the E-step is to compute the cur-

rent expected value of Q with respect to Θ while the subsequent M-step is to maximize Q in Θ^*. These two steps are alternated iteratively until a stopping criterion is satisfied. In the case of missing data, the log-likelihood function is a linear function in the sufficient marginals [7]. The log-likelihood function $l(\Theta \,|\, D)$ of the parameters Θ given the data D and DAG G is:

$$l(\Theta \,|\, D) = \sum_{i=1}^{N} \log P(c_i \,|\, \Theta).$$

In the case of Bayesian networks, the E-step of the EM algorithm is to compute expected counts for each family $\mathrm{fa}(X_i)$ and parent $\mathrm{pa}(X_i)$ configuration of each node X_i under Θ:

$$n^*(Y) = \mathbb{E}_{\Theta}\{n(Y) \,|\, D\},$$

where Y is either $\mathrm{pa}(X_i) = j$ or $X_i = k, \mathrm{pa}(X_i) = j$. The M-step computes new estimates of θ^*_{ijk} from the expected counts under θ_{ijk}:

$$\theta^*_{ijk} = \frac{n^*(X_i = k, \mathrm{pa}(X_i) = j)}{n^*(\mathrm{pa}(X_i) = j)}.$$

The E-step and M-step are iterated until convergence of $l(\Theta)$ (or until a limit on the number of iterations is reached). In the Hugin Tool convergence is achieved when the difference between the log-likelihoods of two consecutive iterations is less than or equal to the numerical value of a log-likelihood threshold times the log-likelihood. Alternatively, the user can specify an upper limit on the number of iterations to ensure that the procedure terminates.

When both data and domain expert knowledge is available, these two sources of knowledge can be fused. In [11] the notion of experience is introduced. Experience is the quantitative knowledge related to a probability distribution based on quantitative expert knowledge. Expert knowledge on the parameters is specified as Dirichlet distributions. For each variable X_i, the distribution $P(X_i \mid \mathrm{pa}(X_i)) = \{p_{ijk}\}$ and the experience counts $\alpha_{i1}, \ldots, \alpha_{i|\,\mathrm{pa}(X_i)|}$ associated with X_i are used to specify the prior expert knowledge. Hence, the experience table of a variable X_i indicates the experience related to the child distribution for each configuration of the parents. In the case of expert knowledge, the E-step does not change whereas the M-step becomes:

$$\theta^*_{ijk} = \frac{n^*(X_i = k, \mathrm{pa}(X) = j) + p_{ijk}\alpha_{ij}}{n^*(\mathrm{pa}(X) = j) + \alpha_{ij}}.$$

The quality of the model is expressed in the value of $l(\Theta \,|\, D)$ computed after each iteration. It should be noticed that $l(\Theta \,|\, D)$ as a quality measure does not incorporate the complexity of the model. For comparison of models with different complexity other measures such as BIC or AIC should be used.

The experience counts for the prior beliefs in the conditional probability distribution of variable X_i given its parents $\mathrm{pa}(X_i)$ are specified in a separate

table including one experience count for each configuration of $\mathrm{pa}(X_i)$. After the termination of the EM algorithm the expected counts are stored as the experience counts.

Example 3.2 [Parameter estimation in Chest Clinic]
Assume that the qualitative knowledge of the Chest Clinic example is as shown in Fig. 1 and that a database $D = \{c_1, \ldots, c_N\}$ with $N = 10,000$ is available. From the qualitative knowledge, we know $E = L \vee T$. This is specified in $P(E \mid L, T)$ and no experience table is allocated to E in order to avoid estimation of this table from the data whereas all other variables have an experience consisting of zeros indicating no expert knowledge on the distributions.

The EM algorithm will produce a maximum likelihood estimate of the parameters of the model under the constraint that $P(E \mid L, T)$ encodes disjunction. If we had expert knowledge on $P(S)$, for instance, we would specify this in $P(S)$ and encode the second order uncertainty in the experience table of S.

3.3 Sequential Updating

Sequential updating or adaptation [11,3] is the task of sequentially updating the conditional probability distributions of a Bayesian network when the structure and an initial specification of the conditional probability distributions are given in advance. In sequential learning, experience is extended to include both quantitative expert knowledge and past cases (e.g. from EM learning). Thus, the result of EM learning could be used as the input for sequential learning.

Let X_i be a variable with n states, then the prior belief in the parameter vector $\theta_{ij} = (\theta_{ij1}, \ldots, \theta_{ijn})$, i.e. the conditional probability distribution of a variable X_i given its parents $\mathrm{pa}(X_i) = j$, is specified as an n-dimensional Dirichlet distribution $\mathcal{D}(\alpha_{ij1}, \ldots, \alpha_{ijn})$. This distribution is represented using a single experience count α_{ij} (equivalent sample size) and the initial content of $P(X_i \mid \mathrm{pa}(X_i) = j)$. The experience count α_{ijk} for a particular state k of X_i given $\mathrm{pa}(X_i) = j$ is $\alpha_{ijk} = \alpha_{ij} p_{ijk}$.

After a complete observation on $(X_i = k, \mathrm{pa}(X_i) = j)$, the posterior belief in the distribution is updated as $\alpha^*_{ijk} = \alpha_{ijk} + 1$ and $\alpha^*_{ijl} = \alpha_{ijl}$ for $l \neq k$. After an incomplete observation, the posterior belief in θ_{ij} is a Dirichlet mixture, which is approximated by a single Dirichlet distribution having the same means and sum of variances as the mixture. The approximation is used in order to avoid the combinatorial explosion, which would otherwise occur when subsequent incomplete observations are made. The updated mean and variance are computed as:

$$m^*_{ijk} = \frac{\alpha_{ijk} + p_{ijk} + m_{ijk}(1 - p_{ijk})}{\alpha_{ij} + 1},$$

$$v^*_{ijk} = \frac{m_{ijk}(1 - m_{ijk})}{\alpha_{ij} + 1}.$$

The updated experience count is computed from the mean and variance.

This process is referred to as retrieval of experience. Dissemination of experience is the process of calculating prior conditional probability distributions for the variables in the Bayesian network given the experience, and it proceeds by setting the value of each parameter equal to the mean of the corresponding updated Dirichlet distribution, i.e. $\theta_{ijk} = m^*_{ijk}$.

In order to reduce the influence of the past and possibly outdated information, an optional feature of fading is provided. Fading proceeds by reducing the experience count before the retrieval of experience takes place. The experience count α_{ij} is faded by a factor of $0 < \lambda_{ij} \leq 1$ typically close to 1 according to $p_{ij} = P(\text{pa}(X_i) = j)$ such that $\alpha^*_{ij} = \alpha_{ij}((1 - p_{ij}) + \lambda_{ij}p_{ij})$. Notice, that experience counts corresponding to parent configurations, which are inconsistent with the evidence are unchanged. The fading factors of a variable X_i are specified in a separate table including one fading factor for each configuration of $\text{pa}(X_i)$.

Example 3.3 [Adaptation in Chest Clinic]
Assume we have evidence $\epsilon = \{S = n, A = y, D = y\}$ *on a patient, i.e. a non-smoking patient with dyspnoea who has recently been to Asia. The evidence is entered and propagated followed by an adaptation of parameters. Table 1 shows the experience counts for* L, B, *and* S *before (i.e. after EM learning using* $10,000$ *randomly generated cases) and after the adaptation with fading factor of* 0.999 *for each distribution. Notice, that since* S *is an observed variable without parents, the experience count* α_S *for* $P(S)$ *will converge to* $\frac{1}{\lambda} = 1001$ *if* $S = n$ *is observed multiple times.*

Table 1. Experience counts for B, L, and S before and after adaptation

	α_S	$\alpha_{L\|S=no}$	$\alpha_{L\|S=yes}$	$\alpha_{B\|S=no}$	$\alpha_{B\|S=yes}$
Before	10,000	4970.88	5029.12	4970.88	5029.12
After	9,001	4472.71	5029.12	4473.73	5029.12

4 Learning Wizard

The learning functionality of the Hugin Tool is supported through a Learning Wizard. A full learning cycle, as performed by the Learning Wizard consists of three main steps: Data acquisition, structural learning, and parameter estimation. Each of these consists of a number of sub-steps, which guide the user in the process of learning the Bayesian network from data and possibly expert knowledge. The user has the option of performing only one of the steps, but in both cases, the data acquisition step is required.

4.1 Data Acquisition

The data acquisition step serves two purposes: Read data from a data source and preprocess the data. In the first step, the user can read in data from various data sources, including data bases and data files. In the second step, the user can preprocess the data, e.g. discretize a variable. It is also possible for the user to use his own preprocessor, if the existing preprocessor does not suffice.

4.2 Structural Learning and Parameter Estimation

The structural learning step contains two sub-steps: Structural learning and data analysis. In the structural learning phase, the user can choose from two algorithms for performing the learning (PC and NPC). Common for these algorithms are, that the user can control the result to some extent by specifying a significance-level parameter and by adding structural constraints on the structure of the DAG before the learning takes place. These structural constraints provide a way for the user to force known dependences/independences onto the learning algorithm. As it can be a tiresome task to specify these constraints for complex networks, the wizard facilitates the saving and loading of network information, including constraints, node positions, node labels, etc.

If the user chooses NPC for the learning algorithm, he will also have the possibility of resolving ambiguous regions or unresolved directions found during the learning process, see e.g. Fig. 1. In the data analysis phase, the strength of both the marginal dependences and the found data dependences can be examined and the complexity of the learned network is indicated.

The parameter estimation phase gives the user the possibility of specifying the initial value of the parameters and the parameters for the EM-algorithm. The initial distribution is determined by any prior possibilities and experience counts specified by the user. To examine if the algorithm may have found a local maximum, it is possible to randomize the prior probabilities, so that the initial distribution can be different for subsequent runs.

5 Performance Evaluation

In the performance evaluation we have used the ALARM network [2], which has become a standard benchmark for structural learning. The ALARM network consists of 37 variables and 46 edges. Each variable has between two and four states with an average of 1.2 parents of each variable.

The PDAG shown in Fig. 3, which is the result of NPC learning on a sample of $10,000$ cases generated from the ALARM [2] network with a significance level $\alpha = 0.01$, contains three ambiguous regions. The three ambiguous regions will be resolved by selecting the *correct* edges ((ArtCO2,Catechol), (VentLung,KinkedTube), and (LVFailure,LVEDVolume)) and adjacent edges are directed correctly. A few edges cannot be directed based on the data alone, a wrong collider is present at Intubation, no other edge is directed incorrectly, no

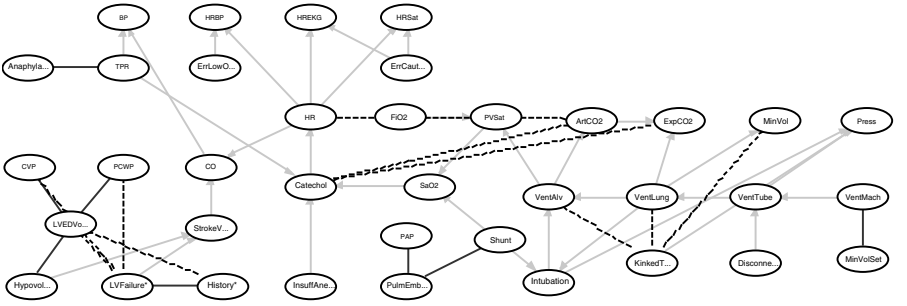


Fig. 3. NPC learning on a sample from ALARM with $\alpha = 0.01$

extra edges are present, and one edge (MinVol,Intubation) is missing. The result of applying the PC algorithm to the same set of data produced a DAG with two incorrect colliders, the two edges (TPR,Anaphylaxis) and (LVFailure,History) missing, and a few incorrect directions on edges. At the moment, the PC algorithm generates a DAG structure where some edges have been given direction at random. Using the NPC algorithm with $\alpha = 0.05$ produce no missing edges, but an additional collider at Intubation.

To evaluate the performance of the PC and NPC algorithms as a function of the significance level α we have performed tests with values of α equal to 0.001, 0.01, 0.05, and 0.1. The results are shown in Table 2. The tests have been performed using samples generated from the ALARM network.

The table shows the number of edges found including neighbors with the number of incorrect edges found in parentheses, the number of edges with correct orientation, the time to perform the learning in milliseconds for both algorithms. Furthermore, for the NPC algorithm the number of ambiguous regions and the number of uncertain edges in each region with the number of missing edges which are represented as an uncertain edge in parentheses. The values are average values over 25 samples of 10,000 cases with 5% missing values (MCAR).

Table 2. Results from using different values of α

Algorithm	α	Edges	Direction	Time (ms)	Regions	Uncertain edges
PC	0.001	45.25(0.5)	44.75	419		
PC	0.01	45.5(0.25)	42.5	426		
PC	0.05	44.25(0)	41.75	415		
PC	0.1	45.25(0.25)	44	434		
NPC	0.001	43.75(0)	39	4,015	1.5	5(1.25)
NPC	0.01	44(0.25)	34.5	4,152	2	7(2)
NPC	0.05	43(0)	36.25	3,805	2.25	9.25(2.75)
NPC	0.1	44.25(0)	38.75	4,125	1	4(1)

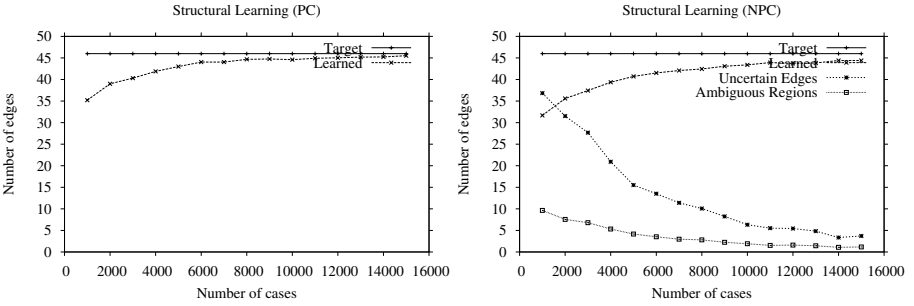


Fig. 4. Results of structural learning as a function of N

Table 3. Average run-time in seconds as a function of number of nodes

	25	50	75	100	150	200
PC	0.2	0.8	2	5	16	29
NPC	2	11	30	71	204	330

The results show that the average run-time of the PC algorithm is lower than that of the NPC algorithm. The PC algorithm is faster since fever tests are performed and since there is no notion of ambiguous regions requiring additional computations. The PC algorithm is able to direct more edges then the NPC algorithm, but some of these are directed at random in order to obtain a DAG.

Some of the differences between the NPC and PC algorithm, which seems to be shortcomings of the NPC algorithm can be remedied by improving the implementation. For instance, the principle of Occam's Razor has not been applied to the ambiguous regions to reduce the number of uncertain edges in each region. On the ALARM network, this led to ambiguous regions containing a single edge, which is present in the ALARM network.

The performances of the PC and NPC algorithms on large networks have been evaluated using randomly generated networks. For a fix size in terms of the number of variables, 10 networks with random topology (zero to five parents) and distribution have been generated. Each variable has from two to five states. The results are shown in Table 3. The time performance tests have been performed using $10,000$ cases with 5% missing cases (MCAR) drawn at random from the distribution of the network.

All tests have been performed on a HP Omnibook xe4500 with a 1700 MHz Pentium 4 processor and 256MB of RAM running Linux Redhat 8.

Demo

A free demo-version of the Hugin Tool can be downloaded from our web-site: `http://www.hugin.com`. Questions related to the functionality of the Hugin Tool can be directed to support@hugin.com.

References

1. S.K. Andersen, K.G. Olesen, F.V. Jensen, and F. Jensen. HUGIN — a Shell for Building Bayesian Belief Universes for Expert Systems. In *Proc. of the 11th IJCAI*, pages 1080–1085, 1989.
2. I.A. Beinlich, H.J. Suermondt, R.M. Chavez, and G.F. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proc. of the Second European Conference on Artificial Intelligence in Medicine*, pages 247–256, 1989.
3. R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, 1999.
4. D. Geiger, T. Verma, and J. Pearl. Identifying independence in Bayesian networks. *Networks*, 20(5):507–534, 1990. Special Issue on Influence Diagrams.
5. D. Heckerman. A tutorial on learning Bayesian networks. In *Learning in Graphical Models*, 1999.
6. F. Jensen, U.B. Kjærulff, M. Lang, and A.L. Madsen. HUGIN - The Tool for Bayesian Networks and Influence Diagrams. In *First European Workshop on Probabilistic Graphical Models*, pages 212–221, 2002.
7. S.L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics & Analysis*, 19:191–201, 1995.
8. C. Meek. Causal inference and causal explanation with background knowledge. In *Proc. of the 11th UAI*, pages 403–410, 1995.
9. J. Pearl. *Probabilistic Reasoning in Intelligence Systems*. Series in Representation and Reasoning. Morgan Kaufmann Publishers, 1988.
10. J. Pearl. *Causality. Models, Reasoning, and Inference*. Cambridge University Press, 2000.
11. D. Spiegelhalter and S.L. Lauritzen. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20:579–605, 1990.
12. P. Spirtes and C. Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computing Review*, 9(1):62–72, 1991.
13. P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Massachusetts, second edition, 2000.
14. H. Steck and V. Tresp. Bayesian belief networks for data mining. Proc. 2nd Workshop "Data Mining und Data Warehousing als Grundlage moderner enscheidungsunterstuetzender Systeme", 1999.
15. T. Verma and J. Pearl. An Algorithm for Deciding if a Set of Observed Independencies Has a Causal Explanation. In *Proc. of the 8th UAI*, pages 323–330, 1992.
16. N. Wermuth and S.L. Lauritzen. Graphical and recursive models for contingency tables. *Biometrika*, 70:537–552, 1983.