

BNT Structure Learning Package : Documentation and Experiments

Philippe Leray, Olivier Francois

TECHNICAL REPORT

Laboratoire PSI - INSA Rouen- FRE CNRS 2645
BP 08 - Av. de l'Université, 76801 St-Etienne du Rouvray Cedex
{Philippe.Leray, Olivier.Francois}@insa-rouen.fr

15th November 2004, Version 1.3



Abstract

Bayesian networks are a formalism for probabilistic reasoning that is more and more used for classification task in data-mining. In some situations, the network structure is given by an expert, otherwise, retrieving it from a database is a NP-hard problem, notably because of the search space complexity. In the last decade, lot of methods have been introduced to learn the network structure automatically, by simplifying the search space (augmented naive bayes, K2) or by using an heuristic in this search space (greedy search). Most of these methods deal with completely observed data, but some others can deal with incomplete data (SEM, MWST-EM).

The Bayes Net Toolbox introduced by [Murphy, 2001a] for Matlab allows us using Bayesian Networks or learning them. But this toolbox is not 'state of the art' if we want to perform a Structural Learning, that's why we propose this package.

Keywords

Bayesian Networks, Structure Learning, Classification.

1 Introduction

Bayesian networks are probabilistic graphical models introduced by [Kim & Pearl, 1987], [Lauritzen & Spiegelhalter, 1988], [Jensen, 1996], [Jordan, 1998].

Definition 1. $\mathcal{B} = (\mathcal{G}, \theta)$ is a bayesian network if $\mathcal{G} = (X, E)$ is a directed acyclic graph (DAG) where the set of nodes represents a set of random variables $X = \{X_1, \dots, X_n\}$, and if $\theta_i = [\mathbb{P}(X_i/X_{Pa(X_i)})]$ is the matrix containing the conditional probability of node i given the state of its parents $Pa(X_i)$.

A bayesian network \mathcal{B} represents a probability distribution over X which admits the following joint distribution decomposition:

$$\mathbb{P}(X_1, X_2, \dots, X_n) = \prod_{i=1}^n \mathbb{P}(X_i/X_{Pa(X_i)}) \quad (1)$$

This decomposition allows the creation of some powerful inference algorithms for which bayesian networks became simple modeling and reasoning tools when the situation is uncertain or the data are incomplete. Then bayesian networks are useful for classification problems when interactions between features can be modeled by conditional probability. When the network structure is not given (by an expert), it is possible to learn it from data. This learning task is hard, because of the search space complexity.

Many softwares deal with bayesian networks, for instance :

- Hugin [Andersen *et al.*, 1989]
- Netica [Norsys, 2003]
- Bayesia Lab [Munteanu *et al.*, 2001]
- TETRAD [Scheines *et al.*, 1994]
- DEAL [Böttcher & Dethlefsen, 2003]
- LibB
- the Matlab Bayes Net Toolbox [Murphy, 2001a]

For our experiments, we use Matlab with the Bayes Net Toolbox [Murphy, 2001a] and the *Structure Learning Package* we developpe and propose on our site [Leray *et al.*, 2003].

This paper is organised as follows. We introduce some general concepts concerning bayesian network structures, how to evaluate these structures and some interesting scoring function properties. In section 3, we describe the common methods to perform structure learning, from causality search to heuristic searches in the bayesian network space, and we discuss of initialisation problems of such methods. In section 4, we compare these methods using two series of tests. In the first serie, we want to retrieve a known structure, and in the other one, we want to obtain a good bayesian network for classification tasks. We then conclude on advantages and drawbacks of such methods, and discuss of future relevant research.

We describe the syntaxe of a function as it follows.

	Ver
<pre>[out1, out2] = function(in1, in2)</pre> <p>brief description of the function.</p> <p>Ver in the top-right corner specify the function location : BNT if it is a native function of the BNT toolbox, or v1.3 if it can be find it in the latest version of the package</p> <p>The following fields are optionnals :</p> <p>INPUTS :</p> <ul style="list-style-type: none"> in1 - description of the input argument in1 in2 - description of the input argument in2 <p>OUTPUTS :</p> <ul style="list-style-type: none"> out1 - description of the output argument out1 out2 - description of the output argument out2 <p>e.g., out = function(in), a sample of the calling syntaxe.</p>	

2 Preliminaries

2.1 Exhaustive search and score decomposability

The first (but naive) idea to find the best network structure is the exploration of all possible graphs in order to choose the graph with the best score. Robinson [Robinson, 1977] has proved that $r(n)$, the number of different structures for a bayesian network with n nodes, is given by the recurrence formula of equation 2.

$$r(n) = \sum_{i=1}^n (-1)^{i+1} \binom{n}{i} 2^{i(n-i)} r(n-i) = n^{2^{\Theta(n)}} \quad (2)$$

This equation gives $r(2) = 3$, $r(3) = 25$, $r(5) = 29281$, $r(10) \simeq 4,2 \times 10^{18}$.

	BNT
<pre>Gs = mk_all_dags(n, order)</pre> <p>generates all DAGs with n nodes according to the optional ordering</p>	

Since equation 2 is super exponential, it is impossible to perform an exhaustive search in a decent time as soon as the node number exceeds 7 or 8. So, structure learning methods often use search heuristics.

In order to explore the DAGs space, we use operators like *arc-insertion* or *arc-deletion*. In order to make this search effective, we have to use a local score to limit the computation to the score variation between two neighbor DAGs.

Definition 2. A score S is said decomposable if it can be writen as the sum or the product of functions that depend only of one vertex and its parents. If n is the graph numbers of vertices, a

decomposable score S must be the sum of local scores s :

$$S(\mathcal{B}) = \sum_{i=1}^n s(X_i, pa(X_i)) \quad \text{or} \quad S(\mathcal{B}) = \prod_{i=1}^n s(X_i, pa(X_i))$$

2.2 Markov equivalent set and Completed-PDAGs

Definition 3. Two DAGs are said equivalent (noted \equiv) if they imply the same set of conditional dependencies (i.e. they have the same joint distribution). The Markov equivalent classes set (named \mathcal{E}) is defined as $\mathcal{E} = \mathcal{A} / \equiv$ where we named \mathcal{A} the DAGs' set.

Definition 4. An arc is said reversible if its reversion lead to a graph which is equivalent to the first one. The space of Completed-PDAGs (CPDAGs or also named essential graphs) is defined as the set of Partially Directed Acyclic Graphs (PDAGs) that have only undirected arcs and unreversible directed arcs.

For instance, as the Bayes' rule gives

$$\mathbb{P}(A, B, C) = \mathbb{P}(A)\mathbb{P}(B|A)\mathbb{P}(C|B) = \mathbb{P}(A|B)\mathbb{P}(B)\mathbb{P}(C|B) = \mathbb{P}(A|B)\mathbb{P}(B|C)\mathbb{P}(C)$$

those structures, $\textcircled{A} \rightarrow \textcircled{B} \rightarrow \textcircled{C} \equiv \textcircled{A} \leftarrow \textcircled{B} \rightarrow \textcircled{C} \equiv \textcircled{A} \leftarrow \textcircled{B} \leftarrow \textcircled{C}$, are equivalents (they all imply $A \perp C|B$).

Then, they can be shematized by the CPDAG $\textcircled{A} \text{---} \textcircled{B} \text{---} \textcircled{C}$ without ambiguities.

But they are not equivalent to $\textcircled{A} \rightarrow \textcircled{B} \leftarrow \textcircled{C}$ (where $\mathbb{P}(A, B, C) = \mathbb{P}(A)\mathbb{P}(B|A, C)\mathbb{P}(C)$) for which the correspondent CPDAG is the same graph which is named a **V-structure**.

[Verma & Pearl, 1990] have proved that DAGs are *equivalent* if, and only if, they have the same skeleton (i.e. the same edge support) and the same set of V-structures (like $\textcircled{A} \rightarrow \textcircled{C} \leftarrow \textcircled{B}$). Furthermore, we make the analogy between the Markov equivalence classes set (\mathcal{E}) and the Completed-PDAGs' set as they are in a natural one-to-one relationship.

[Dor & Tarsi, 1992] propose a method to construct a consistent extension of a DAG.

v1.3

```
dag = pdag_to_dag(pdag)
```

gives an instantiation of a PDAG in the DAG space whereas it is possible.

[Chickering, 1996] introduces a method to find a DAG which instantiate a CDPAG and also proposes the method which permits to find the CDPAG which modelises the equivalence classe of a DAG.

v1.3

```
cpdag = dag_to_cpdag(dag)
```

gives the complete PDAG of a DAG (also works with a cell array of cpdags, returning a cell array of dags).

v1.3

```
dag = cpdag_to_dag(cpdag)
```

gives an instantiation of a CPDAG in the DAG space (also works with a cell array of cpdags, returning a cell array of dags).

2.3 Score equivalence and dimensionality

Definition 5. A score is said equivalent if it gives the same results for equivalent DAGs.

For instance, the BIC score is *decomposable* and *equivalent*. It is derived from principles stated in [Schwartz, 1978] and has the following formulation:

$$BIC(\mathcal{B}, D) = \log \mathbb{P}(D|\mathcal{B}, \theta^{ML}) - \frac{1}{2} Dim(\mathcal{B}) \log N \quad (3)$$

where D is the dataset, θ^{ML} are the parameter values obtained by *likelihood maximisation*, and where the network dimension $Dim(\mathcal{B})$ is defined as follows.

As we need $r_i - 1$ parameters to describe the conditional probability distribution $\mathbb{P}(X_i/Pa(X_i) = pa_i)$, where r_i is the size of X_i and pa_i a specific value of X_i parents, we need $Dim(X_i, \mathcal{B})$ parameters to describe $\mathbb{P}(X_i/Pa(X_i))$ with

$$Dim(X_i, \mathcal{B}) = (r_i - 1)q_i \quad \text{where} \quad q_i = \prod_{X_j \in Pa(X_i)} r_j \quad (4)$$

And the bayesian network dimension $Dim(\mathcal{B})$ is defined by

$$Dim(\mathcal{B}) = \sum_{i=1}^n Dim(X_i, \mathcal{B}) \quad (5)$$

v1.3

```
D = compute_bnet_nparams(bnet)
```

```
gives the number of parameters of the bayesian network bnet
```

The BIC-score is the sum of a likelihood term and a penalty term which penalise complex networks. As two equivalent graphs have the same likelihood and the same complexity, the BIC-score is *equivalent*.

Using scores with these properties, it becomes possible to perform structure learning in Markov equivalent space (*i.e.* $\mathcal{E} = \mathcal{A}/\equiv$). This space has good properties: since a algorithm with

a score on DAGs space can cycle on equivalent networks, the same method with the same score on the \mathcal{E} space will progress (in practice such a method will manipulate CPDAGs).

v1.3

```
score = score_dags(Data, ns, G)
```

compute the score ('bayesian' by default or 'BIC' score) of a dag G
This function exists in BNT, but the new version available in the Structure Package uses a cache to avoid recomputing all the local score in the score_family sub-function when we compute a new global score.

INPUTS :

Data{i,m} - value of node i in case m (can be a cell array).
ns(i) - size of node i.
dags{g} - g'th dag

The following optional arguments can be specified in the form of ('name',value) pairs : [default value in brackets]

scoring_fn - 'bayesian' or 'bic' ['bayesian'] currently,
only networks with all tabular nodes support bayesian scoring.
type - type{i} is the type of CPD to use for node i, where the type is a string of the form 'tabular', 'noisy_or', 'gaussian', etc.
[all cells contain 'tabular']
params - params{i} contains optional arguments passed to the CPD constructor for node i, or [] if none.
[all cells contain {'prior', 1}, meaning use uniform Dirichlet priors]
discrete - the list of discrete nodes [1:N]
clamped - clamped(i,m) = 1 if node i is clamped in case m
[zeros(N, ncases)]
cache - data structure used to memorize local score computations
(cf. SCORE_INIT_CACHE function) [[]]

OUTPUT :

score(g) is the score of the i'th dag

e.g., score = score_dags(Data, ns, mk_all_dags(n), 'scoring_fn', 'bic', 'params', [], 'cache', cache);

In particular CPDAGs can be scored by

v1.3

```
score = score_dags(Data, ns, cpdag_to_dag(CPDAGs), 'scoring_fn', 'bic')
```

As the global score of a DAG is the the product (or the summation is our case as we take the logarithm) of local scores. It can be judicious not forgetting the computed local scores. We can do it by using a cache matrix.

v1.3

```
cache = score_init_cache(N,S);
```

INPUTS:

N - the number of nodes
S - the length of the cache

OUTPUT:

cache - entries are the parent set, the son node, the score of the family, and the scoring method

2.4 discretization

Most structure learning implementations work only with **tabular nodes**. Then the package propose a function to perform discretization. This function gives an optimal discretization proposed in [O.Colot & El Matouat, 1994].

v1.3

```
[n,edges,nbedges,xechan] = hist_ic(ContData,crit)
```

Optimal Histogram based on IC information criterion bins the elements of ContData into an optimal number of bins according to a cost function based on Akaike's Criterion.

INPUTS:

ContData(m,i) - case m for the node i
crit - different penalty terms (1,2,3) for AIC criterion or can ask the function to return the initial histogram (4) [3]

OUTPUTS:

n - cell array containing the distribution of each column of X
edges - cell array containing the bin edges of each column of X
nbedges - vector containing the number of bin edges for each column of X
xechan - discretized version of ContData

When the bin `edges` are given, then the discretisation can be directly done.

v1.3

```
[n,xechan] = histc_ic(ContData,edges)
```

Counts the number of values in `ContData` that fall between the elements in the `edges` vector

INPUTS:

`ContData(m,i)` - case `m` for the node `i`

`edges` - cell array containing the bin edges of each column of `X`

OUTPUT:

`n` - cell array containing these counts

`xechan` - discretized version of `ContData`

3 Algorithms and implementation

The algorithms we use in the following experiments are: PC (causality search), MWST (maximum spanning tree), K2 (with two random initialisations), K2+T (K2 with MWST initialisation), K2-T (K2 with MWST *inverse* initialisation), GS (starting with an empty structure), GS+T (GS starting with MWST structure), GES (greedy search in the space of equivalent classes) and SEM (greedy search dealing with missing values, starting with an empty structure). We also use NB (Naive Bayes) and TANB (Tree Augmented Naive Bayes) for classification tasks.

In the following, the term `n` represents the number of nodes of the expected bayesian network and the number of attributes in the dataset `Data`. Then the size of the dataset is `[n,m]` where `m` is the number of cases.

3.1 Dealing with complete data

3.1.1 A causality search algorithm

A statistical test can be used to evaluate the conditional dependances between variables and then use the results to build the network structure.

PC algorithm has been introduced by [Spirtes *et al.*, 2000] ([Pearl & Verma, 1991] also proposed a similar algorithm (IC) at the same time).

These functions already exist in BNT [Murphy, 2001a]. They need an external function to compute conditional independence tests.

We propose to use `cond_indep_chisquare`.

v1.3

```
[CI Chi2] = cond_indep_chisquare(X, Y, S, Data, test, alpha, ns)
```

This boolean function performs either a Pearson's Chi2 Test or a G2 Likelihood Ratio test

INPUTS :

Data - data matrix, n cols * m rows
X - index of variable X in Data matrix
Y - index of variable Y in Data matrix
S - indexes of variables in set S
alpha - significance level [0.01]
test - 'pearson' for Pearson's chi2 test, 'LRT' for G2 test ['LRT']
ns - node size [max(Data')]

OUTPUTS :

CI - test result (1=conditional independency, 0=no)
Chi2 - chi2 value (-1 if not enough data to perform the test -> CI=0)

Remark that this algorithm does not give a DAG but a completed PDAG which only contains unreversible arcs.

BNT

```
PDAG = learn_struct_pdag_pc('cond_indep', n, n-2, Data);
```

INPUTS:

cond_indep - boolean function that performs statistical tests and that can be called as follows : `feval(cond_indep_chisquare, x, y, S, ...)`
n - number of node
k - upper bound on the fan-in
Data{i,m} - value of node i in case m (can be a cell array).

OUTPUT :

PDAG is an adjacency matrix, in which
PDAG(i,j) = -1 if there is an i->j edge
PDAG(i,j) = P(j,i) = 1 if there is an undirected edge i <-> j

Then to have a DAG, the following operation is needed :

```
DAG = cpdag_to_dag(PDAG);
```

The IC* algorithm learns a latent structure associated with a set of observed variables. The latent structure revealed is the projection in which every latent variable is

- 1) a root node
- 2) linked to exactly two observed variables.

Latent variables in the projection are represented using a bidirectional graph, and thus remain implicit.

BNT
<pre>PDAG = learn_struct_pdag_ic_star('cond_indep_chisquare', n, n-2, Data);</pre> <p>INPUTS:</p> <ul style="list-style-type: none"> cond_indep - boolean function that performs statistical tests and that can be called as follows : feval(cond_indep_chisquare, x, y, S, ...) n - number of node k - upper bound on the fan-in Data{i,m} - value of node i in case m (can be a cell array). <p>OUTPUTS :</p> <ul style="list-style-type: none"> PDAG is an adjacency matrix, in which <ul style="list-style-type: none"> PDAG(i,j) = -1 if there is either a latent variable L such that i <-L-> j OR there is a directed edge from i->j. PDAG(i,j) = -2 if there is a marked directed i->j edge. PDAG(i,j) = PDAG(j,i) = 1 if there is an undirected edge i-j PDAG(i,j) = PDAG(j,i) = 2 if there is a latent variable L such that i<-L->j.

A recent improvement of PC named BN-PC-B has been introduced by [Cheng *et al.*, 2002].

v1.3
<pre>DAG = learn_struct_bnpc(Data);</pre> <p>The following arguments (in this order) are optional:</p> <ul style="list-style-type: none"> ns - a vector containing the nodes sizes [max(Data')] epsilon - value used for the probabilistic tests [0.05] mwst - 1 to use learn_struct_mwst instead of Phase_1 [0] star - 1 to use try_to_separate_B_star instead of try_to_separate_B, more accurate but more complex [0]

3.1.2 Maximum weight spanning tree

[Chow & Liu, 1968] have proposed a method derived from the *maximum weight spanning tree* algorithm (MWST). This method associates a weight to each edge. This weight can be either the *mutual information* between the two variables [Chow & Liu, 1968] or the score variation when one node becomes a parent of the other [Heckerman *et al.*, 1994]. When the weight matrix is

created, an usual MWST algorithm (Kruskal or Prim's ones) gives an undirected tree that can be oriented with the choice of a root.

v1.3

```
T = learn_struct_mwst(Data, discrete, ns, node_type, score, root);
```

INPUTS:

- Data(i,m) is the node i in the case m,
- discrete - 1 if discret-node 0 if not
- ns - arity of nodes (1 if gaussian node)
- node_type - tabular or gaussian
- score - BIC or mutual_info (only tabular nodes)
- root - root-node of the result tree T

OUTPUT:

- T - a sparse matrix that represents the result tree

3.1.3 Naive bayes structure and augmented naive bayes

The naive bayes classifier is a well known classifier related to bayesian networks. Its structure contains only edges from the class node C to the other observations in order to simplify the joint distribution as $\mathbb{P}(C, X_1, \dots, X_n) = \mathbb{P}(C)\mathbb{P}(X_1|C)\dots\mathbb{P}(X_n|C)$

v1.3

```
DAG = mk_naive_struct(n,C)
```

where n is the number of nodes and C the class node

The naive bayes structure suppose that observations are independant given the class, but this hypothesis can be overide by using a *augmented naive bayes* classifier [Keogh & Pazzani, 1999, Friedman *et al.*, 1997a]. We use more precisely a tree augmented structure, where the best tree relying all the observations is obtained with MWST algorithm [Geiger, 1992].

v1.3

```
DAG = learn_struct_tan(Data, C, root, ns, scoring_fn);
```

INPUTS :

- Data - data(i,m) is the m^{st} observation of node i
- C - number of the class node
- root - root of the tree built on the observation node (root \neq C)
- ns - vector containing the size of nodes, 1 if gaussian node
- scoring_fn - (optional) 'bic' (default value) or 'mutual_info'

OUTPUT:

- DAG - TAN structure

3.1.4 K2 algorithm

The main idea of the K2 algorithm is to maximise the structure probability given the data. To compute this probability, we can use the fact that:

$$\frac{\mathbb{P}(\mathcal{G}_1/D)}{\mathbb{P}(\mathcal{G}_2/D)} = \frac{\frac{\mathbb{P}(\mathcal{G}_1, D)}{P(D)}}{\frac{\mathbb{P}(\mathcal{G}_2, D)}{P(D)}} = \frac{\mathbb{P}(\mathcal{G}_1, D)}{\mathbb{P}(\mathcal{G}_2, D)}$$

and the following result given by [Cooper & Hersovits, 1992] :

Theorem 1 *let D the dataset, N the number of examples, and \mathcal{G} the network structure on X . If pa_{ij} is the j^{th} instantiation of $Pa(X_i)$, N_{ijk} the number of data where X_i has the value x_{ik} and $Pa(X_i)$ is instantiated in pa_{ij} and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$ then*

$$\mathbb{P}(\mathcal{G}, D) = \mathbb{P}(\mathcal{G})\mathbb{P}(D|\mathcal{G}) \quad \text{with} \quad \mathbb{P}(D|\mathcal{G}) = \prod_{i=1}^n \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk!} \quad (6)$$

where $\mathbb{P}(\mathcal{G})$ is the prior probability of the structure \mathcal{G} .

Equation 6 can be interpreted as a quality measure of the network given the data and is named the *bayesian measure*.

Given an uniform prior on structures, the quality of a node X and its parent set can be evaluated by the local score described in equation 7.

$$s(X_i, Pa(X_i)) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk!} \quad (7)$$

We can reduce the size of the search space using a node order [Cooper & Hersovits, 1992]. According to this order, a node can be parent only of node which are behind it in this order. The search space becomes the subspace of all the DAGs admitting this order as topological order.

The K2 algorithm tests parent insertion according to a specific order. The first node can't have any parent, and for other nodes, we choose the parents set (within admissible ones) that best upgrade the score.

[Heckerman *et al.*, 1994] has proved that the *bayesian measure* is not *equivalent* and has proposed the BDe score (*bayesian measure* with a specific prior on parameters) to avoid this.

It is also possible to use the BIC score or the MDL score [Bouckaert, 1993] in the K2 algorithm which are both *score equivalent*.

BNT

```
DAG = learn_struct_k2(Data, ns, order);
```

INPUTS:

- Data - Data(i,m) = value of node i in case m (can be a cell array)
- ns - ns(i) is the size of node i
- order - order(i) is the i'th node in the topological ordering

The following optional arguments can be specified in the form of ('name',value) pairs : [default value in brackets]

- max_fan_in - this the largest number of parents we allow per node [N]
- scoring_fn - 'bayesian' or 'bic', currently, only networks with all tabular nodes support Bayesian scoring ['bayesian']
- type - type{i} is the type of CPD to use for node i, where the type is a string of the form 'tabular', 'noisy_or', 'gaussian', etc. [all cells contain 'tabular']
- params - params{i} contains optional arguments passed to the CPD constructor for node i, or [] if none. [all cells contain 'prior', 1, meaning use uniform Dirichlet priors]
- discrete - the list of discrete nodes [1:N]
- clamped - clamped(i,m) = 1 if node i is clamped in case m [zeros(N, ncases)]
- verbose - 'yes' means display output while running ['no']

OUTPUT:

- DAG - The learned DAG which respect with the enumeration order

e.g., dag = learn_struct_K2(data,ns,order,'scoring_fn','bic','params',[])

3.1.5 Markov Chain Monte Carlo

We can use a Markov Chain Monte Carlo (MCMC) algorithm called Metropolis-Hastings (MH) to search the space of all DAGs [Murphy, 2001b]. The basic idea is to use MH algorithm to draw samples from $\mathbb{P}(\mathbf{D}|\mathcal{G})$ (cf equ. 6) after a burn-in time. Then a new graph \mathcal{G}' is keep if

a uniform variable take a value greater than the bayes factor $\frac{\mathbb{P}(\mathbf{D}|\mathcal{G}')}{\mathbb{P}(\mathbf{D}|\mathcal{G})}$ (or a ponderated bayes factor). Remark that this method is not deterministic.

BNT
<pre>[sampled_graphs, accept_ratio, num_edges] = learn_struct_mcmc(Data, ns);</pre> <p>Monte Carlo Markov Chain search over DAGs assuming fully observed data (modified by Sonia Leach)</p> <p>INPUTS:</p> <p> Data - Data(i,m) = value of node i in case m (can be a cell array) ns - ns(i) is the size of node i</p> <p>The following optional arguments can be specified in the form of (‘name’,value) pairs : [default value in brackets]</p> <p> scoring_fn - ‘bayesian’ or ‘bic’, currently, only networks with all tabular nodes support Bayesian scoring [‘bayesian’]</p> <p> type - type{i} is the type of CPD to use for node i, where the type is a string of the form ‘tabular’, ‘noisy_or’, ‘gaussian’, etc. [all cells contain ‘tabular’]</p> <p> params - params{i} contains optional arguments passed to the CPD constructor for node i, or [] if none. [all cells contain ‘prior’, 1, meaning use uniform Dirichlet priors]</p> <p> discrete - the list of discrete nodes [1:N]</p> <p> clamped - clamped(i,m) = 1 if node i is clamped in case m [zeros(N, ncases)]</p> <p> nsamples - number of samples to draw from the chain after burn-in 100*N]</p> <p> burnin - number of steps to take before drawing samples [5*N]</p> <p> init_dag - starting point for the search [zeros(N,N)]</p> <p>OUTPUT:</p> <p> sampled_graphsm = the m’th sampled graph accept_ratio(t) = acceptance ratio at iteration t num_edges(t) = number of edges in model at iteration t</p> <p>e.g., samples = learn_struct_mcmc(data, ns, ‘nsamples’, 1000);</p>

3.1.6 Greedy search

The greedy search is a well-known optimisation heuristic. It takes an initial graph, defines a neighborhood, computes a score for every graph in this neighborhood, and choose the one which maximises the score for the next iteration.

With bayesian networks, we can define the neighborhood as the set of graphs that differ only with one insertion, one reversion or one deletion from our current graph.

As this method is complex in computing time, it is recommended to use a cache.

v1.3

```
DAG = learn_struct_gs2(Data, ns, seeddag, 'cache', cache);
```

This is an improvement of `learn_struct_gs` which was written by Gang Li. As this algorithm computes the score for every graphs in the neighborhood (created with `mk_nbrs_of_dag_topo` developed by Wei Hu instead of `mk_nbrs_of_dag`), we have to use a *decomposable score* to make this computation efficient and then recover some local scores in cache.

INPUT:

Data - training data, `data(i,m)` is the `m` observation of node `i`
ns - the size array of different nodes
seeddag - initial DAG of the search, optional
cache - data structure used to memorize local score computations

OUTPUT:

DAG - the final structure matrix

3.1.7 Greedy search in the Markov equivalent space

Recent works have shown the interest of searching in the Markov equivalent space (see definition 3). [Munteanu & Bendou, 2002] have proved that a greedy search in this space (with an equivalent score) is more likely to converge than in the DAGs space. These concepts have been implemented by [Chickering, 2002a, Castelo & Kocka, 2002, Auvray & Wehenkel, 2002] in new structure learning methods. [Chickering, 2002b] has proposed the *Greedy Equivalent Search* (GES) which used CPDAGs to *represent* Markov equivalent classes. This method works in two phases, first it starts with an empty graph and add arcs until the score cannot be improved, and then it try to suppress some irrelevant arcs.

v1.3

```
DAG = learn_struct_ges(Data, ns, 'scoring_fn', 'bic', 'cache', cache);
```

Like most of others methods, this function can simply be calling as `learn_struct_ges(Data, ns)` but this calling does not take advantages of the caching implementation.

INPUTS:

Data - training data, `data(i,m)` is the `m` observation of node `i`
ns - the size vector of different nodes

The following optional arguments can be specified in the form of ('name',value) pairs : [default value in brackets]

cache - data structure used to memorize local scores [[]]
scoring_fn - 'bayesian' or 'bic' ['bayesian']
verbose - to display learning information ['no']

OUTPUT:

DAG - the final structure matrix

3.1.8 Initialisation problems

Most of the previous methods have some initialisation problems. For instance, the K2 algorithm depends of its enumeration order. As [Heckerman *et al.*, 1994] propose, we can use the oriented tree obtained with the MWST algorithm to generate this order. We just have to initialise the MWST algorithm with a root node, which can be the class node (like in our tests) or randomly chosen. Then we can use the topological order of the tree in order to initialise K2. Let us name "K2+T", the algorithm using this order with the class node as root.

v1.3

```
dag = learn_struct_mwst(Data, ones(n,1), ns, node_type, 'mutual_info', class);
order = topological_sort(full(dag));
dag = learn_struct_K2(Data, ns, order);
```

With this order, where the class node is the root node of the tree, the class node can be interpreting as a cause instead of a consequence. That's why we also propose to use the inverse order, and then name this method "K2-T".

v1.3

```
dag = learn_struct_mwst(Data, ones(n,1), ns, node_type, 'mutual_info', class);
order = topological_sort(full(dag)); order = order(n:-1:1)
dag = learn_struct_K2(Data, ns, order);
```

Greedy search can also be initialised with a specific DAG. If this DAG is not given by an expert, we also propose to use the tree given by the MSWT algorithm to initialise the greedy search instead of an empty network and name this algorithm "GS+T".

v1.3

```
seeddag = full(learn_struct_mwst(Data, ones(n,1), ns, node_type));
cache = score_init_cache(n,cache_size);
dag = learn_struct_gs2(Data, ns, seeddag, 'cache', cache);
```

3.2 Dealing with incomplete data

3.2.1 Structural-EM algorithm

Friedman [Friedman, 1998] first introduced this method for structure learning with incomplete data. This method is based on the *Expectation-Maximisation* principle [Dempster *et al.*, 1977] and deals with incomplete data without adding a new modality to each node which is not fully observed.

This is an iterative method, which convergence has been proved by [Friedman, 1998]. It starts with an initial structure and estimates the probability distribution of missing variables

with the EM algorithm. Then it computes the expectation of the score for each graph of the neighborhood and choose the one which maximises the score.

BNT

```
bnet = learn_struct_EM(bnet, Data, max_loop);
```

INPUTS:

bnet - this function magnipulate the bayesian network bnet instead of only a *DAG* as it learns the parameters in each iteration
Data - training data, data(i,m) is the m obsevation of node i
max_loop - as this method has a big complexity, the maximum loop number must be specify

OUTPUT:

DAG - the final structure matrix

3.2.2 MWST-EM algorithm

under construction ...

4 Experimentation

4.1 Retrieving a known structure

4.1.1 Test networks and evaluation techniques

We used two well-known network structures. The first, ASIA, was introduced by Lauritzen and Spiegelhalter [Lauritzen & Speigelhalter, 1988] (cf figure 1.a). All its nodes are binary nodes. We can notice that concerning the edge between *A* et *T*, the *a priori* probability of *A* is tiny, and the influence of *A* on *T* is weak. The second network we use is INSURANCE with 27 nodes (cf figure 1.b) and is available at [Friedman *et al.*, 1997b].

Data generation has been performed for many sample sizes in order to test the sample size influence on structure learning methods. To generate a sample, we draw the parent node values randomly and choose the son node values according to the bayesian network parameters. These datasets are also randomly cleared of 20% of their values to test the SEM algorithm. Remark that this algorithm is equivalent of the greedy search when the dataset is complete.

In order to compare results obtained with the different algorithms we tested, we use an 'editing measure' defined by the length of the minimal sequence of operators needed to transform the original graph into the resulting one (operators are edge-insertion, edge-deletion and edge-reversal, note that the edge-reversal is considered as a independent operator and not as the deletion and insertion of the opposite edge).

The BIC score of networks is also precised in a compative way (computed from additional datasets of 30000 cases for ASIA and 20000 cases for INSURANCE).

4.1.2 Results and interpretations

Dataset length influence

Figure 2 shows us that MWST algorithm appears to be quite insensitive to dataset length. It always gives a graph close to the original one, although the search space is the tree space which is poorer than the DAGs-space .

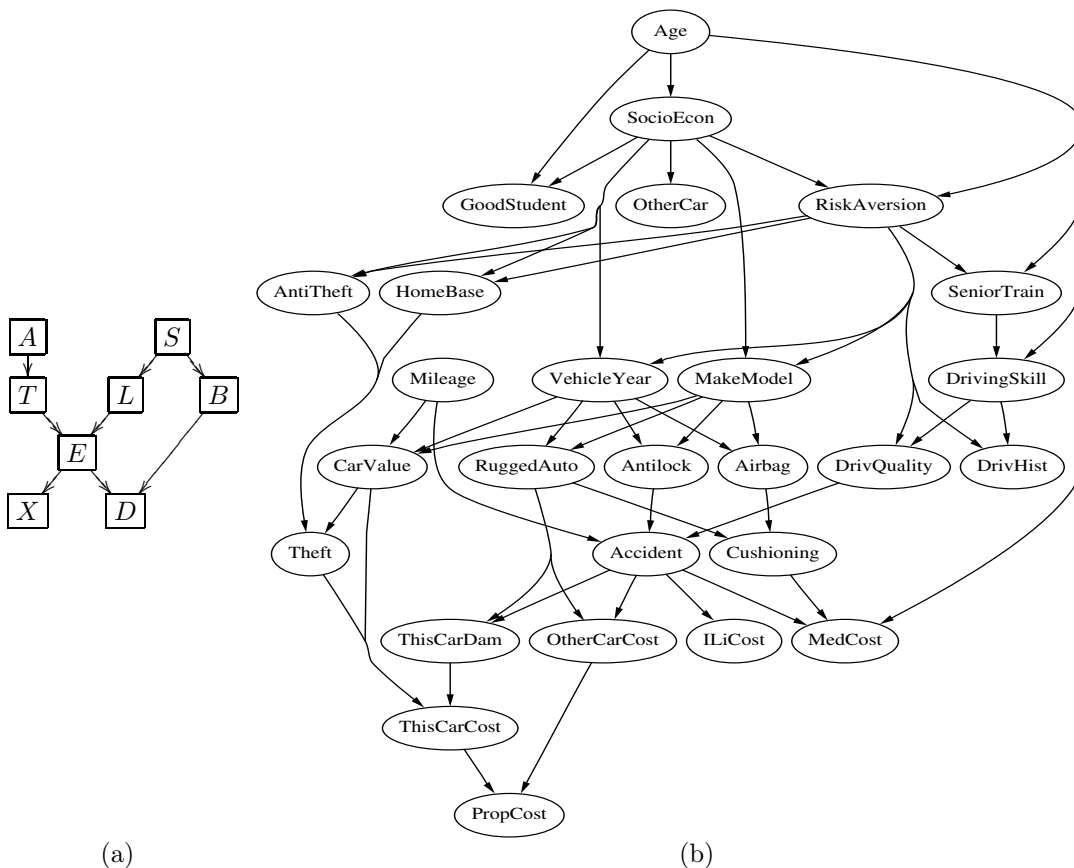


Figure 1: Original networks : (a) ASIA and (b) INSURANCE

The PC also gives good results with a small number of *wrong* edges.

The K2 method is very fast and is frequently use in the literature but have the drawback to be very sensitive to its enumeration order. Figure 2 gives the results of this method on ASIA data with 2 different orders ("ELBXASDT" and "TALDSXEB"). We can notice that K2 results are constant for a given initialisation order, but two different initialisation orders will lead to very different solutions. This phenomenon can also be observed in figure 3 with INSURANCE data.

The results given by BNPC algorithm are good in arc retrieval but do not have great scores.¹

The MCMC based method permit to obtain good results whatever the dataset length, in all runs this method has given similar results from a point of view of the score but they was significant difference for the editing distance.

The GS algorithm is robust to dataset length variation, specially when this algorithm is initialised with MWST tree.

The GES method has given good results whatever the datasets length. With an important amount of data, the networks issued from this methods have greater scores than those found by a classical greedy search. But for the more complicated problem INSURANCE, the results are

¹As this method performs statistical tests it can retrieve dependances that cannot be modelised by a dag then the last step which consists or orienting edges cannot be performed systematicaly (maybe this problem is due to our actual implementation).

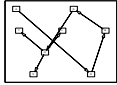
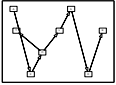
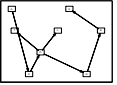
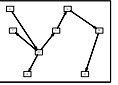
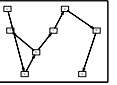
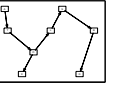
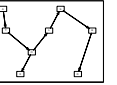
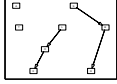
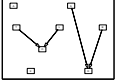
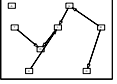
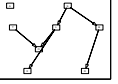
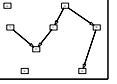
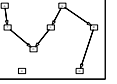
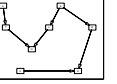





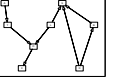
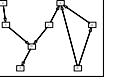


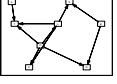


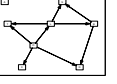
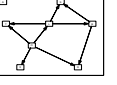


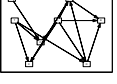
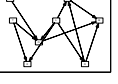

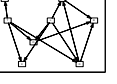
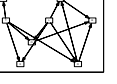
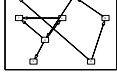
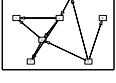
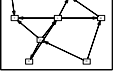
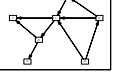
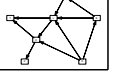
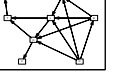
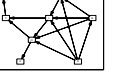










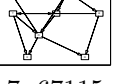
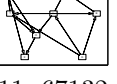

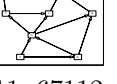
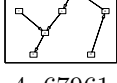
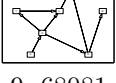
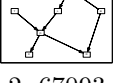
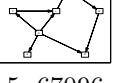
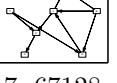

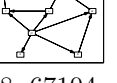
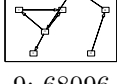
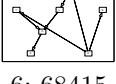
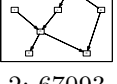
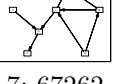
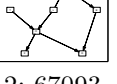
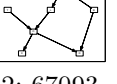
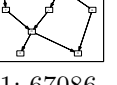
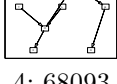
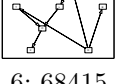
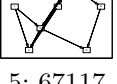
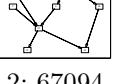
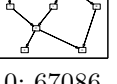
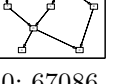
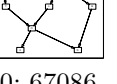

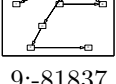
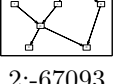
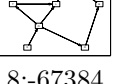
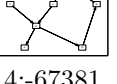

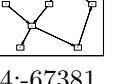
ASIA	250	500	1000	2000	5000	10000	15000
MWST	 9;-68837	 10;-69235	 8;-68772	 6;-68704	 7;-68704	 3;-68694	 3;-68694
PC	 8;-55765	 7;-66374	 6;-61536	 7;-56386	 6;-63967	 5;-63959	 6;-70154
BNPC	 5;-72798	 5;-68492	 5;-72516	 4;-67961	 4;-67964	 5;-68023	 5;-68023
K2	 8;-68141	 7;-67150	 6;-67152	 6;-67147	 6;-67106	 6;-67106	 6;-67106
K2(2)	 11;-68643	 11;-68089	 11;-67221	 10;-67216;	 9;-67129	 9;-67129	 9;-67129
K2+T	 10;-68100	 8;-68418	 9;-67185	 8;-67317	 8;-67236	 10;-67132	 10;-67132
K2-T	 7;-68097	 6;-67099	 6;-67112	 7;-67105	 6;-67091	 5;-67091	 5;-67091
MCMC	 12;-67338	 11;-68428	 9;-67103	 7;-67115	 11;-67132	 11;-67107	 11;-67112
GS	 4;-67961	 9;-68081	 2;-67093	 5;-67096	 7;-67128	 9;-67132	 8;-67104
GS+T	 9;-68096	 6;-68415	 2;-67093	 7;-67262	 2;-67093	 2;-67093	 1;-67086
GES	 4;-68093	 6;-68415	 5;-67117	 2;-67094	 0;-67086	 0;-67086	 0;-67086
SEM	 10;-83615	 9;-81837	 2;-67093	 8;-67384	 4;-67381	 5;-67108	 4;-67381

Figure 2: Editing measures, networks and BIC scores obtained with different methods (in row) for several dataset lengths (in column).

INSURANCE	250	500	1000	2000	5000	10000	15000
MWST	37;-3373	34;-3369	36;-3371	35;-3369	34;-3369	34;-3369	34;-3369
K2	56;-3258	62;-3143	60;-3079	64;-3095	78;-3092	82;-3080	85;-3085
K2(2)	26;-3113	22;-2887	20;-2841	21;-2873	21;-2916	18;-2904	22;-2910
K2+T	42;-3207	40;-3009	42;-3089	44;-2980	47;-2987	51;-2986	54;-2996
K2-T	55;-3298	57;-3075	57;-3066	65;-3007	70;-2975	72;-2968	73;-2967
MCMC*	50;-3188	44;-2967	46;-2929	40;-2882	50;-2905	51;-2898	54;-2892
GS	37;-3228	39;-3108	30;-2944	33;-2888	29;-2859	25;-2837	28;-2825
GS+T	43;-3255	35;-3074	28;-2960	26;-2906	33;-2878	19;-2828	21;-2820
GES	43;-2910	41;-2891	39;-2955	41;-2898	38;-2761	38;-2761	38;-2752
SEM	50;-4431	57;-4262	61;-4396	61;-4092	69;-4173	63;-4105	63;-3978

Figure 3: Editing measures and BIC scores divided by 100 and rounded obtained with different methods (in row) for several dataset lengths (in column) (* As the method MCMC is not deterministic the results are a mean over five runs).

quite better for the scoring function and are worse for the editing distance than those obtained with a greedy search in the DAGs space.

Whatever the dataset length, SEM method always gives identical results on ASIA data and very similar results on INSURANCE data. Notice that this method obtains bad editing measure because it retrieves a bad oriented ASIA structure. Automatically distinguishing if a bad orientation is a real mistake (by breaking a V-structure for instance) or not is difficult. We are currently working on an editing distance that takes into account this problem by working into Markov equivalent classes.

Weak dependance recovering

Most of the tested methods have not recovered the $A-T$ edge of the ASIA structure. Only the simple method MWST, PC and K2 initialised with MWST structure retrieve this edge when the dataset is big enough. This can be explained for all the scoring methods: this edge-insertion does not lead to a score increase because the likelihood increase is counterbalanced by the penalty term increase.

4.2 Learning Efficient Bayesian Network for Classification

4.2.1 Datasets and evaluation criterion

ASIA

We reuse the dataset previously generated with 2000 instances for the learning phase and the one with 1000 instances for testing.

HEART

This dataset, available from Statlog project [Sutherland & Henery, 1992, Michie *et al.*, 1994], is a medical diagnosis dataset with 14 attributes (continuous attributes have been discretised). This dataset has 270 data we decompose into 189 learning data and 81 test data.

AUSTRALIAN

This dataset, which is available on [Michie *et al.*, 1994], consists in a credit offer evaluation granted to a Australian customer evaluate considering 14 attributes. It contains 690 cases which have been separated into 500 instances for learning and 190 for testing.

LETTER

This dataset from [Michie *et al.*, 1994] is the only one we tested which have not a binary classification but the arity of the classe is 26. It has been created from handwritten letter recognition and contains 16 attributes like position or height of a letter but also means or variances of the pixels in x and in y axis. It contains 15000 samples for learning and 5000 samples for testing.

THYROID

This dataset, available at [Blake & Merz, 1998], is a medical diagnosis dataset. We use 22 attributes (among the 29 original ones): 15 discrete attributes, 6 continuous attributes that have been discretised and one (binary) class node. This dataset has 2800 learning data and 972 test data.

CHESS

This dataset is also available at [Blake & Merz, 1998] (Chess – King+Rook versus King+Pawn). It is a chess prediction task: determining if white can win the game according to the current position described by 36 attributes (the class is the 37th). This dataset has 3196 data we decompose into 2200 learning data and 996 test data.

Evaluation

The evaluation criterion is the good classification percentage on test data, with an $\alpha\%$ confidence interval proposed by [Bennani & Bossaert, 1996] (cf eq. 8).

$$I(\alpha, N) = \frac{T + \frac{Z_\alpha^2}{2N} \pm Z_\alpha \sqrt{\frac{T(1-T)}{N} + \frac{Z_\alpha^2}{4N^2}}}{1 + \frac{Z_\alpha^2}{N}} \quad (8)$$

where N is the sample size, T is the classifier good classification percentage and $Z_\alpha = 1.96$ for $\alpha = 95\%$.

4.2.2 Results and interpretations

Classifier performances and confidence intervals corresponding to several structure learning algorithms are given table 1. These results are compared with a k-nearest-neighbour classifier ($k = 9$).

Let notice that the *memory crash* obtained with PC algorithm on medium size datasets is due to the actual implementation of this method. [Spirtes *et al.*, 2000] propose an heuristic that can be used on bigger datasets than the actual implementation.

For simple classification problems like ASIA, a naive bayes classifier gives as good results as complex algorithms or as the KNN methods. We can also point up that the tree search method (MWST) gives similar or better results than naive bayes for our datasets. It appears judicious to use this simple technic instead of the naive structure. Contrary to our intuition the TANB classifier gives little worse results than the naive bayes classifier except on HEART dataset where the results are much worse and on LETTER problem where it has given the best recognition rate (except if we consider the KNN). Even if this method permit to relax the conditionnal independances between the observations, it also increase the network complexity, and then the number of parameters that we have to estimate is too big for our dataset length.

For more complex problems like CHESS, structure learning algorithms obtain better performances than naive bayes classifier.

Differing to the previous structure search experience, the several initialisations we use with the K2 algorithm do not lead to improve the classification rate. Nevertheless, using another method to choose the initial order permits to stabilize the method.

The MCMC method gives poor results for problem with a small number of nodes but seems to be able to find very good structure when the number of nodes increase.

	ASIA	HEART	AUTRALIAN	LETTER	THYROID	CHESS
att, L, T	8, 2000, 1000	14, 189, 81	15, 500, 190	17, 15000, 5000	22, 2800, 972	37, 2200, 996
NB	86.5% _[84.2;88.5]	87.6% _[78.7;93.2]	87.9% _[82.4;91.8]	73.5% _[72.2;74.7]	95.7% _[94.2;96.9]	86.6% _[84.3;88.6]
TANB	86.5% _[84.2;88.5]	81.5% _[71.6;88.5]	86.3% _[80.7;90.5]	85.3% _[84.3;86.3]	95.4% _[93.8;96.6]	86.4% _[84.0;88.4]
MWST-BIC	86.5% _[84.2;88.5]	86.4% _[77.3;92.3]	87.4% _[81.8;91.4]	74.1% _[72.9;75.4]	96.8% _[95.4;97.8]	89.5% _[87.3;91.3]
MWST-MI	86.5% _[84.2;88.5]	82.7% _[73.0;89.5]	85.8% _[80.1;90.1]	74.9% _[73.6;76.1]	96.1% _[94.6;97.2]	89.5% _[87.3;91.3]
PC	84.6% _[82.2;86.8]	85.2% _[75.7;91.3]	86.3% _[80.7;90.5]	memory crash	memory crash	memory crash
K2	86.5% _[84.2;88.5]	83.9% _[74.4;90.4]	83.7% _[77.8;88.3]	74.9% _[73.6;76.1]	96.3% _[94.9;97.4]	92.8% _[90.9;94.3]
K2+T	86.5% _[84.2;88.5]	81.5% _[71.6;88.5]	84.2% _[78.3;88.8]	74.9% _[73.6;76.1]	96.3% _[94.9;97.4]	92.6% _[90.7;94.1]
K2-T	86.5% _[84.2;88.5]	76.5% _[66.2;84.5]	85.8% _[80.1;90.1]	36.2% _[34.9;37.6]	96.1% _[94.6;97.2]	93.0% _[91.2;94.5]
MCMC*	86.44%	84.20%	80.00%	72.96%	96.17%	95.62%
GS	86.5% _[84.2;88.5]	85.2% _[75.8;91.4]	86.8% _[81.3;91.0]	74.9% _[73.6;76.1]	96.2% _[94.7;97.3]	94.6% _[93.0;95.9]
GS+T	86.2% _[83.9;88.3]	82.7% _[73.0;89.5]	86.3% _[80.7;90.5]	74.9% _[73.6;76.1]	95.9% _[94.4;97.0]	92.8% _[90.9;94.3]
GES	86.5% _[84.2;88.5]	85.2% _[75.8;91.4]	84.2% _[78.3;88.8]	74.9% _[73.6;76.1]	95.9% _[94.4;97.0]	93.0% _[91.2;94.5]
SEM	86.5% _[84.2;88.5]	80.2% _[70.2;87.5]	74.2% _[67.5;80.0]	memory crash	96.2% _[94.7;97.3]	89.2% _[87.1;91.0]
kNN	86.5% _[84.2;88.5]	85.2% _[75.8;91.4]	80.5% _[74.3;85.6]	94.8% _[94.2;95.5]	98.8% _[97.8;99.4]	94.0% _[92.3;95.4]

Table 1: Good classification percentage on test data and 95% confidence interval for classifiers obtained with several structure learning algorithms (Naive Bayes, Tree Augmented Naive Bayes with Mutual Information score, Maximum Weight Spanning Tree with Mutual Information or BIC score, PC, K2 initialisate with [class node , observation nodes with numerous order] or with MWST or *inverse* MWST initialisation, MCMC (* As this method is not deterministic the results are a mean over five runs), Greedy Search starting with an empty graph or with MWST tree, Greedy Equivalent Search and Structural EM dealing with 20% of missing data. These results are compared with a k-nearest-neighbour classifier ($k = 9$).

Surprisingly, the Greedy Search do not find a structure with a better classification rate, although this method walk through the entire DAGs space. It can be explain by the size of the dag space and the great number of local optimum in this.

By a theoretical way, the Greedy Equivalent Search is the most advanced score based method of those we tested. In the preceding experiments, it permits to find structures with great scores. But on our classification problems, the performances are a little weaker than those obtained by a classical greedy search.

On the other hand, Structural EM successfully manages to deal with incomplete datasets and obtains results similar to other methods with 20% of missing data.

The methods we used do not give better results than the k nearest neighbour classifier. But we can notice that the resulting bayesian network can also be use in many ways. For instance by inferring on other nodes than the class one, by interpreting the structure or also by dealing with missing data.

5 Conclusions and future work

Learning bayesian network structure from data is a difficult problem for which we reviewed the main existing methods.

Our first experiment allowed us to evaluate the precision of these methods retrieving a known graph. Results show us that finding weak relation between attributes is difficult when the sample size is too small. For most of the methods, random initialisations can be replaced effectively by initialisations issued from a simple algorithm like MWST.

Our second experiment permitted to evaluate the effectiveness of these methods for classification task. Here, we have shown that a good structure search can lead to results similar to the kNN method but can also be used in other ways (structure interpreting, inference on other nodes) and deal with incomplete data. Moreover simple methods like Naive Bayes or MWST give results as good as more complex methods on simple problems (*i.e. with few nodes*).

Recent works show that walking through the Markov equivalent space (cf definition 3) instead of the DAGs space lead to optimal results. Munteanu *et al.* [Munteanu & Bendou, 2002] proved that this space has better properties and [Chickering, 2002a, Castelo & Kocka, 2002] propose a new structure learning in this space. Moreover [Chickering, 2002a] proved the optimality of his methods GES. In our experiments, this method has permitted to have the best results if we consider the scoring function, but if we consider the editing distance or the classification rate the results are not so satisfying.

Adapting existing methods to deal with missing data is very important to treat real problems. SEM algorithm perform a greedy search in the DAGs space but the same principle could be used with other algorithms (MWST for instance) in order to quickly find a good structure with incomplete data. Some initialisation problems are also to be solved. Finally, the last step will be adapting Structural EM to Markov equivalent search methods.

Acknowledgements

This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

References

- [Andersen *et al.*, 1989] ANDERSEN S., OLESEN K., JENSEN F. & JENSEN F. (1989). Hugin - a shell for building bayesian belief universes for expert systems. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, p. 1080–1085. <http://www.hugin.com/>.
- [Auvray & Wehenkel, 2002] AUVRAY V. & WEHENKEL L. (2002). On the construction of the inclusion boundary neighbourhood for markov equivalence classes of bayesian network structures. In A. DARWICHE & N. FRIEDMAN, Eds., *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI-02)*, p. 26–35, S.F., Cal.: Morgan Kaufmann Publishers.
- [Bennani & Bossaert, 1996] BENNANI Y. & BOSSAERT F. (1996). Predictive neural networks for traffic disturbance detection in the telephone network. In *Proceedings of IMACS-CESA '96*, Lille, France.
- [Blake & Merz, 1998] BLAKE C. & MERZ C. (1998). UCI repository of machine learning databases. <http://www.ics.uci.edu/~mllearn/MLRepository.html>.
- [Bouckaert, 1993] BOUCKAERT R. R. (1993). Probabilistic network construction using the minimum description length principle. *Lecture Notes in Computer Science*, **747**, 41–48.
- [Böttcher & Dethlefsen, 2003] BÖTTCHER S. G. & DETHLEFSEN C. (2003). Deal: A package for learning bayesian networks.

- [Castelo & Kocka, 2002] CASTELO R. & KOCKA T. (2002). *Towards an inclusion driven learning of bayesian networks*. Rapport interne UU-CS-2002-05, Institute of information and computing sciences, University of Utrecht.
- [Cheng *et al.*, 2002] CHENG J., GREINER R., KELLY J., BELL D. & LIU W. (2002). Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, **137**(1–2), 43–90.
- [Chickering, 1996] CHICKERING D. (1996). Learning equivalence classes of Bayesian network structures. In E. HORVITZ & F. JENSEN, Eds., *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)*, p. 150–157, San Francisco: Morgan Kaufmann Publishers.
- [Chickering, 2002a] CHICKERING D. M. (2002a). Learning equivalence classes of bayesian-network structures. *Journal of machine learning research*, **2**, 445–498.
- [Chickering, 2002b] CHICKERING D. M. (2002b). Optimal structure identification with greedy search. *Journal of Machine Learning Research*, **3**, 507–554.
- [Chow & Liu, 1968] CHOW C. & LIU C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, **14**(3), 462–467.
- [Cooper & Hersovits, 1992] COOPER G. & HERSOVITS E. (1992). A bayesian method for the introduction of probabilistic networks from data. *Maching Learning*, **9**, 309–347.
- [Dempster *et al.*, 1977] DEMPSTER A., LAIRD N. & RUBIN D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, **B 39**, 1–38.
- [Dor & Tarsi, 1992] DOR D. & TARSİ M. (1992). *A simple algorithm to construct a consistent extension of a partially oriented graph*. Rapport interne R-185, Cognitive Systems Laboratory, UCLA Computer Science Department.
- [François & Leray, 2004] FRANÇOIS O. & LERAY P. (2004). évaluation d’algorithme d’apprentissage de structure dans les réseaux bayésiens. In *14ieme Congrès francophone de Reconnaissance des formes et d’Intelligence artificielle*, p. 1453–1460: LAAS-CNRS.
- [Friedman, 1998] FRIEDMAN N. (1998). The Bayesian structural EM algorithm. In G. F. COOPER & S. MORAL, Eds., *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, p. 129–138, San Francisco: Morgan Kaufmann.
- [Friedman *et al.*, 1997a] FRIEDMAN N., GEIGER D. & GOLDSZMIDT M. (1997a). Bayesian network classifiers. *Machine Learning*, **29**(2-3), 131–163.
- [Friedman *et al.*, 1997b] FRIEDMAN N., GOLDSZMIDT M., HECKERMAN D. & RUSSELL S. (1997b). Challenge: What is the impact of bayesian networks on learning?, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (NIL-97)*, 10-15. <http://www.cs.huji.ac.il/labs/compbio/Repository/>.
- [Geiger, 1992] GEIGER D. (1992). An entropy-based learning algorithm of bayesian conditional trees. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighth Conference (UAI-1992)*, p. 92–97, San Mateo, CA: Morgan Kaufmann Publishers.

- [Heckerman *et al.*, 1994] HECKERMAN D., GEIGER D. & CHICKERING M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. In R. L. DE MANTARAS & D. POOLE, Eds., *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, p. 293–301, San Francisco, CA, USA: Morgan Kaufmann Publishers.
- [Jensen, 1996] JENSEN F. V. (1996). *An introduction to Bayesian Networks*. Taylor and Francis, London, United Kingdom.
- [Jordan, 1998] JORDAN M. I. (1998). *Learning in Graphical Models*. The Netherlands: Kluwer Academic Publishers.
- [Keogh & Pazzani, 1999] KEOGH E. & PAZZANI M. (1999). Learning augmented bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*, p. 225–230.
- [Kim & Pearl, 1987] KIM J. & PEARL J. (1987). Convice; a conversational inference consolidation engine. *IEEE Trans. on Systems, Man and Cybernetics*, **17**, 120–132.
- [Lauritzen & Spiegelhalter, 1988] LAURITZEN S. & SPEIGELHALTER D. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Royal statistical Society B*, **50**, 157–224.
- [Leray *et al.*, 2003] LERAY P., GUILMINEAU S., NOIZET G., FRANCOIS O., FEASSON E. & MINOC B. (2003). French BNT site. <http://bnt.insa-rouen.fr/>.
- [Michie *et al.*, 1994] MICHIE D., SPIEGELHALTER D. J. & TAYLOR C. C. (1994). *Machine Learning, Neural and Statistical Classification*. <http://www.amsta.leeds.ac.uk/~charles/statlog/>
<http://www.liacc.up.pt/ML/statlog/datasets/>.
- [Munteanu & Bendou, 2002] MUNTEANU P. & BENDOU M. (2002). The eq framework for learning equivalence classes of bayesian networks. In *First IEEE International Conference on Data Mining (IEEE ICDM)*, p. 417–424, San José.
- [Munteanu *et al.*, 2001] MUNTEANU P., JOUFFE L. & WUILLEMIN P. (2001). Bayesia lab.
- [Murphy, 2001a] MURPHY K. (2001a). The BayesNet Toolbox for Matlab, *Computing Science and Statistics: Proceedings of Interface*, 33. <http://www.ai.mit.edu/~murphyk/Software/BNT/bnt.html>.
- [Murphy, 2001b] MURPHY K. P. (2001b). Active learning of causal bayes net structure.
- [Norsys, 2003] NORSYS (2003). Netica.
- [O.Colot & El Matouat, 1994] O.COLOT, C. OLIVIER P. C. & EL MATOUAT A. (1994). Information criteria and abrupt changes in probability laws. *Signal Processing VII : Théorie and Applications*, p. 1855–1858.
- [Pearl & Verma, 1991] PEARL J. & VERMA T. S. (1991). A theory of inferred causation. In J. F. ALLEN, R. FIKES & E. SANDEWALL, Eds., *KR'91: Principles of Knowledge Representation and Reasoning*, p. 441–452, San Mateo, California: Morgan Kaufmann.
- [Robinson, 1977] ROBINSON R. W. (1977). Counting unlabeled acyclic digraphs. In C. H. C. LITTLE, Ed., *Combinatorial Mathematics V*, volume 622 of *Lecture Notes in Mathematics*, p. 28–43, Berlin: Springer.

- [Scheines *et al.*, 1994] SCHEINES R., SPIRITES P., GLYMOUR C. & MEEK C. (1994). *Tetrad ii: Tools for discovery*. Lawrence Erlbaum Associates, Hillsdale, NJ.
- [Schwartz, 1978] SCHWARTZ G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, **6**(2), 461–464.
- [Spirtes *et al.*, 2000] SPIRITES P., GLYMOUR C. & SCHEINES R. (2000). *Causation, Prediction, and Search*. The MIT Press, 2 edition.
- [Sutherland & Henery, 1992] SUTHERLAND A. I. & HENERY R. J. (1992). Statlog - an ESPRIT project for the comparison of statistical and logical learning algorithms. *New Techniques and Technologies for Statistics*.
- [Verma & Pearl, 1990] VERMA T. & PEARL J. (1990). Equivalence and synthesis of causal models. In *in Proceedings Sixth Conference on Uncertainty and Artificial Intelligence*, San Francisco: Morgan Kaufmann.