BNT STRUCTURE LEARNING PACKAGE : Documentation and Experiments

Olivier C.H. FRANCOIS   FRANCOIS.OLIVIER.C.H@GMAIL.COM
Sustainable Urban Environments Research Division, URS building, Room 3n38,
Whiteknights, PO Box 219, Reading, RG6 6AW, UK
http://ofrancois.tuxfamily.org

Philippe LERAY   PHILIPPE.LERAY@UNIV-NANTES.FR
Ecole Polytechnique de l'Université de Nantes,
Laboratoire d'Informatique de Nantes Atlantique,
Rue Christian Pauc, BP 50609, 44306 Nantes Cedex 3
http://www.polytech.univ-nantes.fr/COD/?Pages_personnelles:Philippe_Leray

Editor: ?? Leslie Pack Kaelbling ??

Abstract

Bayesian networks are a formalism for probabilistic reasoning that have grown increasingly popular for tasks such as classification in data-mining. In some situations, the structure of the Bayesian network can be given by an expert. If not, retrieving it automatically from a database of cases is a NP-hard problem; notably because of the complexity of the search space. In the last decade, numerous methods have been introduced to learn the network's structure automatically, by simplifying the search space or by using an heuristic in the search space. Most methods deal with completely observed data, but some can deal with incomplete data. The *Bayes Net Toolbox* for Matlab, introduced by Murphy (2004), offers functions for both using and learning Bayesian Networks. But this toolbox is not 'state of the art' as regards structural learning methods. This is why we propose the SLP package.

Keywords: Bayesian Networks, Structure Learning, Classification

1. Introduction

Bayesian networks are probabilistic graphical models introduced by Kim and Pearl (1987), Lauritzen and Spiegelhalter (1988), Jensen (1996), Jordan (1998).

Definition 1. $\mathcal{B} = (\mathcal{G}, \theta)$ *is a* discrete *bayesian network (or belief network) if* $\mathcal{G} = (X, E)$ *is a directed acyclic graph (*DAG*) where the set of nodes represents a set of random variables* $X = \{X_1, \cdots, X_n\}$*, 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:

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

This decomposition allows the use 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. Bayesian networks are also practical for classification problems when interactions between features can be modelized with conditional probabilities. When the network structure is not given (by an expert), it is possible to learn it automatically from data. This learning task is hard, because of the complexity of the search space. Many softwares deal with Bayesian networks, for instance :

- gR Lauritzen et al. (2004)

- BNT Murphy (2004)

- PNL Bradski (2004)

- BNJ Perry and Stilson (2002)

- TETRAD Spirtes et al. (2004)

- Causal explorer Tsamardinos et al. (2005)

- LibB Friedman and Elidan (1999)

- BNPC Cheng et al. (2001)

- Web WeavR Xiang (1999)

- JavaBayes Drakos and Moore (1998)

- ProBayes Mazer et al. (2004)

- BayesiaLab Munteanu et al. (2001)

- Hugin Andersen et al. (1989)

- Netica Netica (1998)

- BayesWare Sebastiani et al. (1999)

- MSBNx Kadie et al. (2001)

- B-Course Myllymäki et al. (2002)

- Bayes Builder Nijman et al. (2002)

For experiments, we have used Matlab with the Bayes Net Toolbox Murphy (2004) and the *Structure Learning Package* we develop and propose over our website Leray et al. (2003). This paper is organized as follows. We introduce some general concepts concerning Bayesian network structures, how to evaluate these structures and some interesting properties of scoring functions. In section 3, we describe the common methods used in structure learning; from causality search to heuristic searches in the Bayesian network space. We also discuss the initialization problems of such methods. In section 4, we compare these methods using two series of tests. In the first series, we try to retrieve a known structure while the other tests aim at obtaining a good Bayesian network for classification tasks. We then conclude on the respective advantages and drawbacks of each method or family of methods before discussing future relevant research. We describe the syntax of a function as follows.

```
                                                                    Ver
[out1, out2] = function(in1, in2)
Brief description of the function.
'Ver', in the top-right corner, specifies the function location :  BNT if it is a
native function of the BNT, or v1.5 if it can be found in the SLP package
The following fields are optionals :
INPUTS :
    in1 - description of the input argument in1
    in2 - description [default value in brackets for optional arguments]
OUTPUTS :
    out1 - description of the output argument out1

e.g., out = function(in), a sample of the calling syntax.
```

2. Preliminaries

2.1 Exhaustive search and score decomposability

The first (but naive) idea as to finding the best network structure is the exploration and evaluation of all possible graphs in order to choose the best structure. Robinson Robinson (1977) has proven that $r(n)$, the number of different structures for a Bayesian network with n nodes, is given by the recursive formula of equation 2.

$$r(n) = \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{i} 2^{i(n-i)} r(n-i) = n^{2^{\mathcal{O}(n)}} \tag{2}$$

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

BNT

```
Gs = mk_all_dags(n, order)

generates all DAGs with n nodes according to the optional ordering
```

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 neighboring DAGs.

Definition 2. *A score S is said to be* decomposable *if it can be written as the sum or the product of functions that depend only of one vertex and its parents. If n is the numbers of vertices in the graph, a decomposable score S must be the sum of* local scores s:

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

2.2 Markov equivalent set and Completed-PDAGs

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

Definition 4. *An arc is said to be* reversible *if its reversion leads 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 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, $(A) \rightarrow (B) \rightarrow (C) \equiv (A) \leftarrow (B) \rightarrow (C) \equiv (A) \leftarrow (B) \leftarrow (C)$, are equivalent
(they all imply $A \perp\!\!\!\perp C | B$).
Then, they can be schematized by the CPDAG $(A) - (B) - (C)$ without ambiguities.
But they are not equivalent to $(A) \rightarrow (B) \leftarrow (C)$ (where $\mathbb{P}(A, B, C) = \mathbb{P}(A)\mathbb{P}(B|A, C)\mathbb{P}(C)$) for which the corresponding CPDAG is the same graph, which is named a **V-structure**.

3

Verma and Pearl (1990) have proven 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 $(A) \rightarrow (B) \leftarrow (C)$). Furthermore, we make the analogy between the Markov equivalence classes set ($\mathcal{E}$) and the set of Completed-PDAGs as they share a natural one-to-one relationship. Dor and Tarsi (1992) proposes a method to construct a consistent extension of a DAG.

> v1.5
>
> ```
> dag = pdag_to_dag(pdag)
> ```
> gives an instantiation of a PDAG in the DAG space whenever it is possible.

Chickering (1996) introduces a method for finding a DAG which instantiates a CDPAG and also proposes the method which permits to find the CDPAG representing the equivalence classe of a DAG.

> v1.5
>
> ```
> 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).

> v1.5
>
> ```
> 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 to be* equivalent *if it returns the same value for equivalent* DAG*s.*

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

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

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

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

And the dimension of the Bayesian network is defined by $Dim(\mathcal{B}) = \sum_{i=1}^{n} Dim(X_i, \mathcal{B})$.

> v1.5
>
> ```
> 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 penalizes 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 using a score over the DAGs space can happen to 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).

v1.5

```
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 Pack-
age 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 evaluated with

v1.5

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

As the global score of a DAG is the product (or the summation is our case as we take the logarithm) of local scores, caching previously computed local scores can prove to be judicious. We can do so by using a cache matrix.

v1.5

```
cache = score_init_cache(N,S);
INPUTS:
    N - the number of nodes
    S - the lentgh of the cache

OUTPUT:
    cache - entries are the parent set, the son node, the score of the
        familly, and the scoring method
```

2.4 discretization

Most structure learning implementations work solely with tabular nodes. Therefore, the SLP package comprises a discretizing function. This function, proposed in O.Colot and El Matouat (1994), returns an optimal discretization.

v1.5

```
[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, the discretization can be done directly.

v1.5

```
[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 weight spanning tree), K2 (with two random initializations), K2+T (K2 with MWST initialization), K2-T (K2 with MWST *inverse* initialization), GS (starting from an empty structure), GS+T (GS starting from a MWST-initialized structure), GES (greedy search in the space of equivalent classes) and SEM (greedy search dealing with missing values, starting from 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 dependencies between variables and then use the results to build the network structure. The PC algorithm has been introduced by Spirtes et al. (2000). Pearl and Verma (1991) also proposed a similar algorithm (IC) at the same time.
These functions already exist in BNT Murphy (2004). They need an external function to compute conditional independence tests.

```
                                                                    v1.5
[CI Chi2] = cond_indep_chisquare(X, Y, S, Data, test, alpha, ns)

This boolean function perfoms either a Pearson's Chi2 Test or a G2 Likelyhood
Ration 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.

```
PDAG = learn_struct_pdag_pc('cond_indep', n, n-2, Data);
INPUTS:
    cond_indep - boolean function that perfoms 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 a root node or is linked to exactly two observed variables. Latent variables in the projection are represented using a bidirectional graph, and thus remain implicit.

```
PDAG = learn_struct_pdag_ic_star('cond_indep_chisquare', n, n-2, Data);
INPUTS:
    cond_indep - boolean function that perfoms 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).

OUTPUTS :
    PDAG is an adjacency matrix, in which
        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 and undirected edge i-j
        PDAG(i,j) = PDAG(j,i) = 2 if there is a latent variable L such that
            i<-L->j.
```

A improvement of PC, BNPC-B Cheng et al. (2002), has been introduced.

```
DAG = learn_struct_bnpc(Data);
The following arguments (in this order) are optionnal:
    ns - a vector containing the nodes sizes [max(Data')]
    epsilon - value uses 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 complexe [0]
```

3.1.2 MAXIMUM WEIGHT SPANNING TREE

Chow and Liu (1968) have proposed a method derived from the *maximum weight spanning tree* algorithm (MWST). This method associates a weigth to each edge. This weight can be either the *mutual information* between the two variables Chow and 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, a usual MWST algorithm (Kruskal or Prim's ones) gives an undirected tree that can be oriented given a root.

```
                                                                    v1.5
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 $\mathbb{P}(C, X_1, ..., X_n) = \mathbb{P}(C)\mathbb{P}(X_1|C)...\mathbb{P}(X_n|C)$

```
                                                                    v1.5
DAG = mk_naive_struct(n,C)
where n is the number of nodes and C the class node
```

The naive bayes structure supposes that observations are independent given the class, but this hypothesis can be overridden using an *augmented naive bayes* classifier Keogh and Pazzani (1999); Friedman et al. (1997a). Precisely, we use a tree-augmented structure, where the best tree relying all the observations is obtained by the MWST algorithm Geiger (1992).

```
                                                                    v1.5
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≠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 maximize 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 and 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 jth 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 $Nij = \sum_{k=1}^{r_i} N_{ijk}$ then*

$$\mathbb{P}(\mathcal{G},D) = \mathbb{P}(\mathcal{G})\mathbb{P}(D|\mathcal{G}) \quad with \quad \mathbb{P}(D|\mathcal{G}) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i-1)!}{(N_{ij}+r_i-1)!} \prod_{k=1}^{r_i} N_{ijk}! \qquad (5)$$

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

Equation 5 can be interpreted as a quality mesure of the network given the data and is named the *Bayesian mesure*.

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 6.

$$s(X_i, Pa(X_i)) = \prod_{j=1}^{q_i} \frac{(r_i-1)!}{(N_{ij}+r_i-1)!} \prod_{k=1}^{r_i} N_{ijk}! \qquad (6)$$

We can reduce the size of the search space using a topological order over the nodes Cooper and Hersovits (1992). According to this order, a node can only be the parent of lower-ordered nodes. The search space thus becomes the subspace of all the DAGs admitting this very topological order.

The K2 algorithm tests parent insertion according to a specific order. The first node can't have any parent while, as for other nodes, we choose the parents sets (among admitable ones) that leads to the best score upgrade.

Heckerman et al. (1994) has proven that the *Bayesian mesure* is not *equivalent* and has proposed the BDe score (*Bayesian mesure* with a specific prior on parameters) to make it so. It is also possible to use the BIC score or the MDL score Bouckaert (1993) in the K2 algorithm which are both *score equivalent*.

```
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',[])
```

```
[sampled_graphs, accept_ratio, num_edges] = learn_struct_mcmc(Data, ns);
Monte Carlo Markov Chain search over DAGs assuming fully observed data

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
The following optional arguments can be specified as in K2
    scoring_fn, type, params, discrete, clamped,
    nsamples - number of samples to draw from the chain after burn-in 100*N]
    burnin - number of steps to take before drawing samples [5*N]
    init_dag - starting point for the search [zeros(N,N)]

OUTPUT:
    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

e.g., samples = learn_struct_mcmc(data, ns, 'nsamples', 1000);
```

3.1.5 MARKOV CHAIN MONTE CARLO

We can use a Markov Chain Monte Carlo algorithm called Metropolis-Hastings (MH) to search the space of all DAGs Murphy (2001). The basic idea is to use the MH algorithm to draw samples from $\mathbb{P}(\mathbf{D}|\mathcal{G})$ (cf equ. 5) after a burn-in time. Then a new graph $\mathcal{G}'$ is kept 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.

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 chooses 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 by one insertion, reversion or deletion of an arc from our current graph.

As this method is complex in computing time, we recommend to use a cache.

```
                                                                    v1.5
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 (cre-
ated with mk_nbrs_of_dag_topo developped 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 obsevation 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 and 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 and Kocka (2002); Auvray and 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 steps. First, it starts with an empty graph and adds arcs until the score cannot be improved, and then it tries to suppress some irrelevant arcs.

```
                                                                    v1.5
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 obsevation 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 INITIALIZATION PROBLEMS

Most of the previous methods have some initialization problems. For instance, the run of the K2 algorithm depends on the given 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 initialize the MWST algorithm with a root node, which can either be the class node (as in in our tests) or randomly chosen. Then we can use the topological order of the tree in order to initializ K2. Let us name "K2+T", the algorithm using this order with the class node as root.

```
                                                                    v1.5
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 interpreted as a cause instead of a consequence. That's why we also propose to use the reverse order. We name this method "K2-T". Simply replace `order` by `order(n:-1:1)` in the code above.

Greedy search can also be initialized 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 initialize the greedy search instead of an empty network and name this algorithm "GS+T".

```
                                                                    v1.5
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 proven by Friedman (1998). It starts from an initial structure and estimates the probability distribution of variables which data are missing with the EM algorithm. Then it computes the expectation of the score for each graph of the neighborhood and chooses the one which maximises the score.

```
                                                                            BNT
bnet = learn_struct_EM(bnet, Data, max_loop);
INPUTS:
    bnet - this function manipulates the baysesian 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

In the same way than for completly observed datasets, we could use an algorithm based on *minimum spanning tree* search from incomplete datasets. This method gives an tree shaped bayesian network that could be use as it, or that could be use to initialize the `learn_struct_EM` algorithm. Let see Leray and François (2005) for implementation details and experimentations.

```
                                                                           v1.5
bnet = learn_struct_mwst_EM(data, discrete, node_sizes, prior, nbloopmax,
t)
INPUTS :
    datai,m a cell where the node i in the case m,
    discrete = [ 1 if discret-node 0 if not ], (ones)
    node_sizes = 1 if gaussian node, (max on complete samples)
    prior = 1 to use uniform Dirichlet prior (0)
    root is the futur root-node of the tree T. (random)
    nbloopmax = max loop number (ceil(log(N*log(N))))
    t = the convergence test's threshold (1e-3)
OUTPUTS :
    bnet = the output bayesian network
    Ebic = the espected BIC score of bnet given the data
```

3.2.3 TAN-EM algorithm

To learn a bayesian network classifier from incomplete datasets, the function `learn_struct_tan_EM` allow to obtain very good classification rates whatever the size of the dataset with a very small learning time.

> v1.5

```
bnet = learn_struct_tan_EM(data, class, node_sizes, root, prior, nbloop-
max, t)
INPUTS and OUTPUTS are the same than learn_struct_mwst_EM
```

3.2.4 GES-EM algorithm

An extention of `learn_struct_ges` to incomplete datasets is also proposed. This version could use a initial structure that is not an empty one. Remark that the GES algorithm is said to be optimal when it is intitialise with an empty structure. Nevertheless, as the learning time is high an as this extention is no more optimal another choice could be made. Let see Borchani et al. (2006) for experimentations.

> v1.5

```
[bnet,cpdag,BIC_score,nloop] = learn_struct_ges_EM(bnet, data, max_loop,
loop_em)
INPUTS :
    bnet is an initial bayesian network structure,
    datai,m a cell where the node i in the case m,    max_loop is the number of
step in the structure space,
    loop_em is the number of loop of the inner EM on parameters
OUTPUTS :
    bnet is the final result as a bayesian network,
    cpdag is the final partialy oriented structure,
    BIC_score is the score on learning data of bnet,
    nloop is the number of loop that has been used.
```

3.2.5 Using pairwise deletion instead EM

Poeple who does not want to use EM estimation as it is too much time computing could use some complete structure learning algorithms with pairwise deletion estimation (available cases analysis). The `scoring_family` function in the Structure Learning package allow to perform count on incomplete datasets using a cell array as input.

> v1.5

```
A way to implement learn_struct_ges_ACA
if data is an array containing the value 'misv' for missing values :  GESACA =
learn_struct_ges(mat_to_bnt(data,misv,), ns,'scoring_fn','bic','cache',cache))
```

3.3 Generating incomplete data

To generate incomplete data, you first have to build a model, and then you could use it to build the dataset. See François and Leray (2007) for details on modelisation.

3.3.1 MCAR CASE MODEL BUILDING

Methods that have been proposed for MCAR dataset generation usually remove data for each variable with the same probability α. We propose here a more general method where a different "missing" probability is associated to each variable.

In the special case of MCAR mechanisms Rubin (1976), we have $\mathbb{P}(\mathcal{R}|\mathcal{O}, \mathcal{H}, \mu) = \mathbb{P}(\mathcal{R}|\mu)$.

```
                                                                    v1.5
bnet_miss = gener_MCAR_net(bnet_orig, base_proba)
INPUTS :
    bnet_orig :  a bnet,
    base_proba :  a goal mean probability for value to be missing
OUTPUTS :
    bnet_miss :  a bnet that could be used in gener_data_from_bnet_miss function
to generate incomplete MCAR dataset.
```

3.3.2 MAR CASE MODEL BUILDING

For MAR processes, nodes representing the missingness of a variable can no longer be disconnected from observable nodes (we now have $\mathbb{P}(\mathcal{R}|\mathcal{O}, \mathcal{H}, \mu) = \mathbb{P}(\mathcal{R}|\mathcal{O}, \mu)$). So there are more a lot more parameters than in MCAR mechanisms to fix.

```
                                                                    v1.5
bnet_miss = gener_MAR_net(bnet_orig, base_proba)
same INPUTS and OUPUTS as gener_MCAR_net
```

3.3.3 GENERATING DATA

```
                                                                    v1.5
[data, comp_data, bnet_miss, rate, bnet_orig, notok] =
gener_data_from_bnet_miss(bnet_miss, m, base_proba ,v, testdata)
INTPUTS :
    bnet_orig :  see gener_[MCAR or MAR]_net function,
    m :  the length of the dataset
    base_proba :  a goal mean probability for value to be missing
    v==1 to enter the verbose mode [0]
    testdata==1 to always build the same dataset <- rand('state',0) [0]
OUTPUTS :
    data :  the generated dataset
    data :  the full original dataset
    bnet_miss :  see gener_[MCAR or MAR]_net function
    ratem :  the mean rate of missing data in data
    bnet_orig :  the original bnet
    notok==1 iff the generated dataset missing rate is to far from ratem.
```

4. Experimentation

4.1 Retrieving a known structure

TEST NETWORKS AND EVALUATION TECHNIQUES

We used two well-known network structures. The first, ASIA, was introduced by Lauritzen and Spiegelhalter (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 small, 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 in Friedman et al. (1997b).

Data generation has been performed for different sample sizes in order to test the influence of this size over the results of the various 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.

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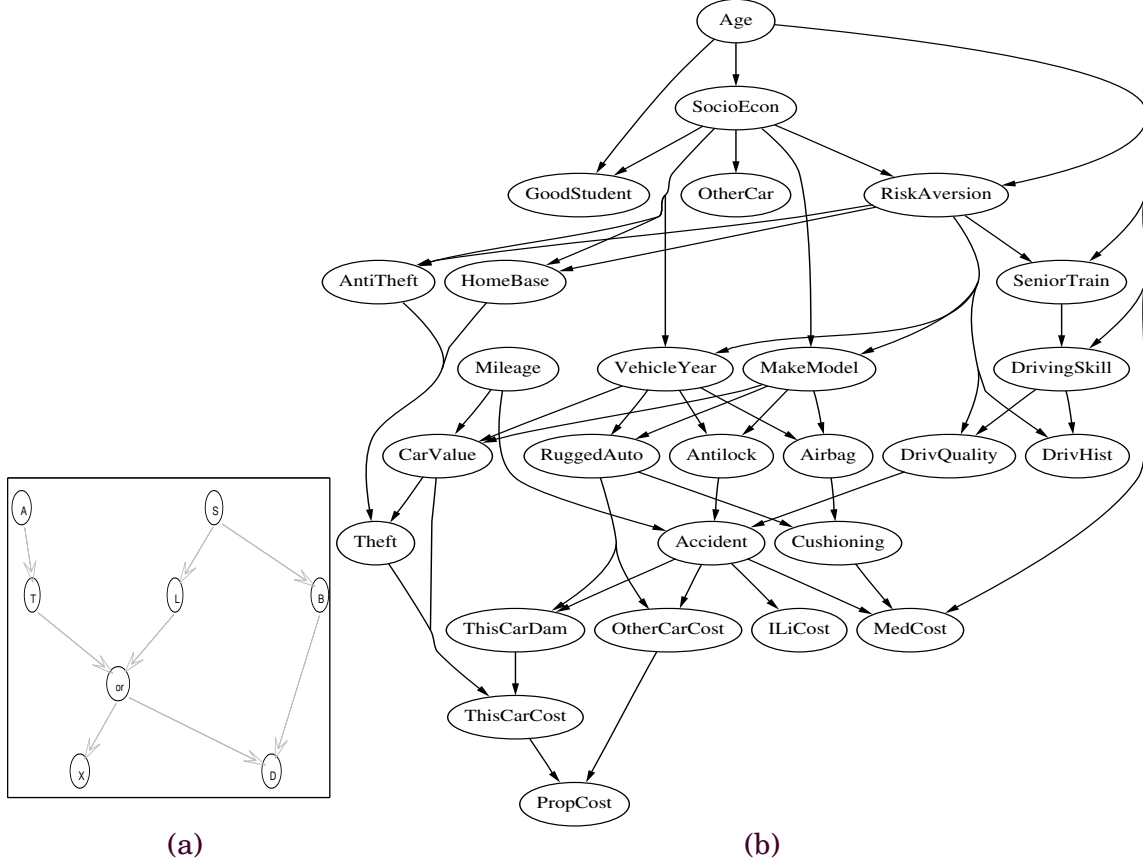
Figure 1: Original networks : (a) ASIA and (b) INSURANCE

In order to compare the results obtained by 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).

	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
BN-PC							
	11;-67825	6;-73885	6;-72529	6;-72529	7;-73141	6;-69046	6;-69370
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
GS-0							
	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

Table 1: Editing measures, networks and BIC scores obtained with different methods (in row) for several dataset lengths (in column).

18

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

Table 2: 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 meaned over five runs).

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

RESULTS AND INTERPRETATIONS

Dataset length influence
Figure 1 shows us that MWST algorithm appears to be quite insensitive to the length of the dataset. 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 .

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

The K2 method is very fast and is frequently used in the literature but presents the drawback of being very sensitive to its initial enumeration order. Figure 1 shows the results of K2 on ASIA data with 2 different orders ("ELBXASDT" and "TALDSXEB"). We can notice that the results are constant for a given initialization order, but two different initialization orders will lead to very different solutions. This phenomenon can also be observed in figure 2 with the INSURANCE data sets.

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

The MCMC based method permit to obtain good results whatever the dataset length. In all runs, this method has given similar results from a scoring point of view but there was significant differences among the editing distances.

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

The GES method has given good results whatever the dataset length. Given an significant amount of data, the networks issued from this method return better scores than those found by a classical greedy search. But for the more complex INSURANCE network, the results are significantly better as for the scoring function than those

obtained with a greedy search in the DAGs space but are worse in terms of editing distances.

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

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 and Henery (1992); Michie et al. (1994), is a medical diagnosis dataset with 14 attributes (continuous attributes have been discretized). This dataset is made of 270 cases which we split into two sets of respectively 189 cases as learning data and 81 cases as test data.

AUSTRALIAN
This dataset, which is available on Michie et al. (1994), consists in a credit offer evaluation granted to an 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 doesn't consist in a binary classificiation: the arity of the class variable being of 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 over the x and the y axis. It contains 15000 samples for learning and 5000 samples for testing.

THYROID
This dataset, available at Blake and 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 is made of 2800 learning data cases and 972 test data cases.

CHESS
This dataset is also available at Blake and 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 is made of 3196 data cases we decompose into 2200 learning data cases and 996 test data cases.

	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$±0.14	84.20%±2.95	80.00%±0	72.96%±4.99	96.17%±0.16	95.62%1.79
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]
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 3: 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 deternimistic the results are meaned over five runs), Greedy Search starting with an empty graph or with MWST tree, Gready Equivalent Search. These results are compared with a k-nearest-neighbour classifier ($k = 9$).

Evaluation

The evaluation criterion is the good classification percentage on test data, with an $\alpha\%$ confidence interval proposed by Bennani and Bossaert (1996) (cf eq. 7).

$$I(\alpha, N) = \frac{T + \frac{Z_\alpha^2}{2N} \pm Z_\alpha \sqrt{\frac{T(1-T)}{N} + \frac{Z_\alpha^2}{4N^2}}}{1 + \frac{Z_\alpha^2}{N}} \tag{7}$$

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

RESULTS AND INTERPRETATIONS

Classifier performances and confidence intervals corresponding to several structure learning algorithms are given table 3. These results are compared with a k-nearest-neighbour classifier ($k = 9$).

Notice that the *memory crash* obtained with PC algorithm on medium-sized datasets is due to the actual implementation of this method. Spirtes et al. (2000) proposes a heuristic that can be used on bigger datasets than the actual implementation can.

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 that 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 permits to relax the conditional independencies between the observations, it also increases 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 an improvement of the classification rate. Nevertheless, using another method to choose the initial order permits to stabilize the method. The MCMC method gives poor results for problems with a small number of nodes but seems to be able to find very good structures as the number of nodes increases. Surprisingly, the Greedy Search does not find a structure with a better classification rate, although this method parses the entire DAGs space. It can be explained by the size of the dag space and the great number of local optima in it. In theory, the Greedy Equivalent Search is the most advanced score based method of those we tested. In the previous experiments, it lead to the finding of high-scoring structures. But over our classification problems, its results are out-performed by those obtained by a classical greedy search.

Bayesian networks outperform the k-nearest neighbor classifier on AUSTRALIAN dataset and kNN outperforms on LETTER dataset. But we can notice that the resulting Bayesian network can also be used in many ways. For instance by infering on other nodes than the class one, by interpretating the structure.

4.3 Retrieving a known structure from incomplete datasets

TEST NETWORKS AND EVALUATION TECHNIQUES

For these experiments, we have used the ASIA network of figure 1.a Lauritzen and Spiegelhalter (1988) to generate full datasets by MCMC simulation of various sizes (500, 1000, 2000, 5000, 10000). These datasets are randomly cleared of the third (33.33%) of their values to test learning algorithms from incomplete datasets. These structure learning algorithms are equivalent to greedy searches in tree space, in DAG space and in CPDAG space.

RESULTS AND INTERPRETATIONS

Results are shown in table 4. First look at figures shows that using pairwise deletion leads to high number of arcs whilst the use of the EM algorithm leads to low number of arcs. These differences don't apply for MWST, and one could see MWST-ACA gives better results than MWST-EM. Note that MWST-ACA is the only 'direct' algorithm, and it is very time efficient for such quality of results.

Differences between GS-ACA and SEM (GS-EM) one one side and GES-ACA and GES-EM on the other side are very close. SEM and GES-EM find too few edges when dataset size is small and tend too give good results for bigger datasets. Surprisingly GS-ACA ang GES-ACA give very good results when we care about the BIC score. Even if the results are very complexe ones, they really capture the distribution of data (without overfitting as BIC score should be low). But structures from these methods could
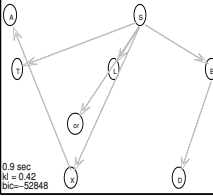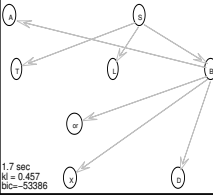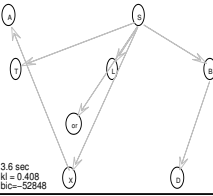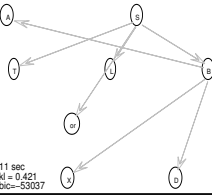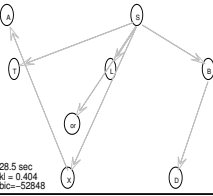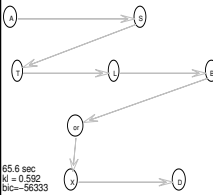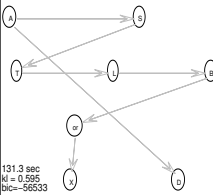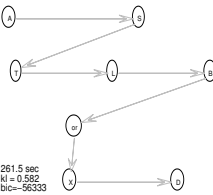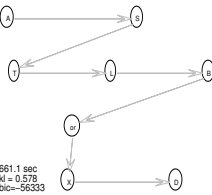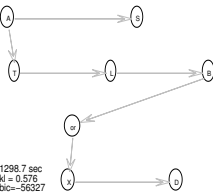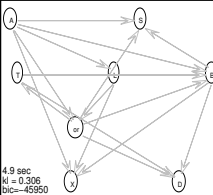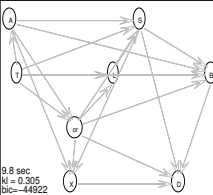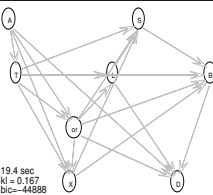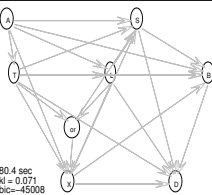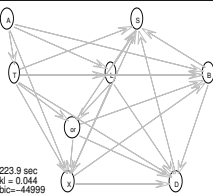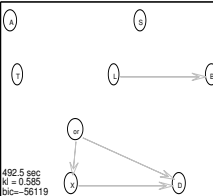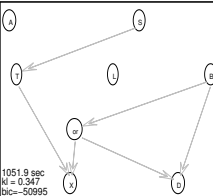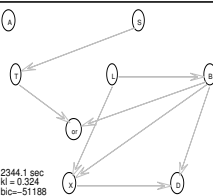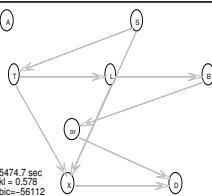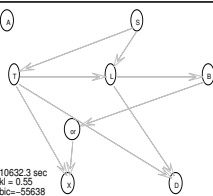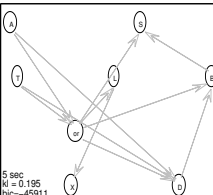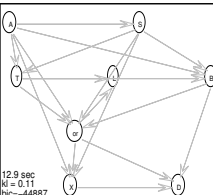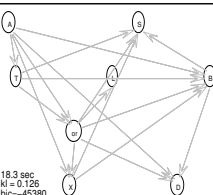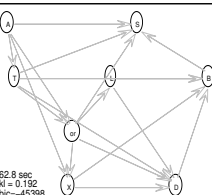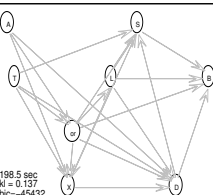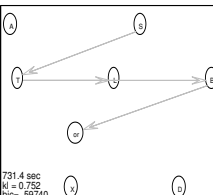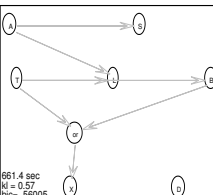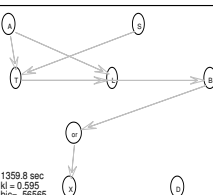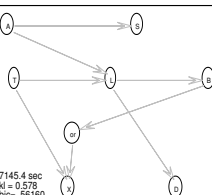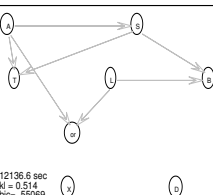
	500	1000	2000	5000	10000
MWST ACA	9; -52848	9; -53386	9; -52848	9; -53037	9; -52848
MWST EM	13; -56333	13; -56533	13; -56333	13; -56333	11; -56327
GS ACA	16; -45950	15; -44922	14; -44888	19; -45008	19; -44999
GS EM	8; -56119	8; -50995	12; -51188	15; -56112	13; -55638
GES ACA	12; -45911	13; -44887	13; -45380	15; -45398	18; -45432
GES EM	12; -59740	11; -56005	11; -56565	14; -56160	9; -55069

Table 4: Editing measures, networks and BIC scores obtained with different methods (in row) for several dataset lengths (in column). Computational time and Kullback-Leiber divergence means (on five parameters learning) are also printed in boxes.

not be interprete as they are too complexe. On the other side, SEM and GES-EM give interpretable structure but with lower scores.

Then results from EM methods are better to understand while results from ACA seems to be better for classification or simulation tasks. In the next section we will see if it is true for classification task.

4.4 Learning Efficient Bayesian Network classifier from incomplete datasets

TEST DATASETS AND EVALUATION TECHNIQUES

The experiment stage aims at evaluating the Tree Augmented Naive Bayes classifier on incomplete datasets from Blake and Merz (1998): Hepatitis, Horse, House, Mushrooms and Thyroid.

The TAN-EM method we proposed here is compared to the Naive Bayes classifier with EM parameters learning. We also indicate the classification rate obtained by three methods: MWST-EM, SEM initialised with a random chain and SEM initialised with the tree given by MWST-EM (SEM+T). The first two methods are dedicated to classification tasks while the others do not consider the class node as a specific variable. We also give an 95%-confidence interval based on equation 7 for each classification rate.

RESULTS AND INTERPRETATIONS

The results are summed up in table 5. First, we could see that even if the Naive Bayes classifier often gives good results, the other tested methods allow to obtain better classification rates. Whislt all runs of NB and ACA methods give same results, EM methods do not always give same results because of the first parameters estimation random initialisation. We have also noticed (not reported here) that TAN-EM seems the most stable method concerning the evaluated classification rate while MWST-EM seems to be the less stable of EM methods.

The method GS-EM could obtain very good structures with a good initialisation. Then, initialising it with the results of MWST-EM gives stabler results (see Leray and François (2005) for a more specific study of this point).

In our tests, except for Hepatits dataset (that have only 90 learning samples), TAN-EM and TAN-ACA always obtain structures that lead to better classification rates in comparison with the other structure learning methods.

Remark that MWST methods could occasionaly give good classification rates even if the class node is connected to a maximum of two other attributes. In that case, it could be a good hint for most relevant attributs to the class node.

Regarding the log-likelihood reported in table 5, we see that GS-ACA give best results while TAN methods finds structures that can also lead to a good approximation of the underlying probability distribution of the data, even with a strong constraint on the graph structure.

In these experiments, we could confirm that ACA methods could outperform EM methods on classification for GS and GES learning methods but not systematicaly. Results are similar for MWST and TAN methods.

Method	HEPATITIS	HORSE	HOUSE	MUSHROOMS	THYROID
sizes	20; 90; 65; 8%	28; 300; 300; 88%	17; 290; 145; 46%	23; 5416; 2708; 31%	22; 2800; 972; 30%
NB	73.8% [62.0;83.0]	73.5% [62.0;82.6]	89.7% [83.6;93.6]	94.4% [93.5;95.2]	96.0% [94.6;97.1]
	-1122 (0s)	-1540 (0s)	-1404 (0s)	-41147 (0s)	-15728 (0s)
MWST-ACA	58.5% [46.3;69.6]	82.4% [71.6;89.6]	90.3% [84.4;94.2]	75.0% [73.3;76.6]	77.4% [74.6;79.9]
	-847 (2s)	-1240 (16s)	-1282 (5s)	-31447 (178s)	-15359 (96s)
MWST-EM	75.4% [63.7;84.2]	82.4% [71.6;89.6]	82.1% [75.0;87.5]	60.3% [58.5;62.2]	93.8% [92.1;95.2]
	-1114 (45s)	-1306 (299s)	-1462 (67s)	-39773 (1389s)	-16912 (2254s)
TAN-ACA	64.6% [52.5;75.1]	73.5% [62.0;82.6]	93.1% [87.8;96.2]	98.4% [97.8;98.8]	95.9% [94.4;97.0]
	-1123 (2s)	-1319 (15s)	-1284 (4s)	-20453 (183s)	-15894 (86s)
TAN-EM	64.6% [52.5;75.1]	77.9% [66.7;86.2]	91.7% [86.1;95.2]	98.4% [97.8;98.8]	97.0% [95.7;97.9]
	-1186 (71s)	-1546 (307s)	-1339 (185s)	-33885 (2345s)	-16292 (1936s)
GS-ACA	67.7% [55.6;77.8]	80.9% [70.0;88.5]	91.7% [86.1;95.2]	76.7% [75.0;78.2]	77.4% [74.6;79.9]
	-865 (55s)	-1052 (774s)	-1289 (71s)	-25256 (9086s)	-15394 (2537s)
SEM	64.6% [52.5;75.1]	51.5% [39.8;62.9]	67.6% [59.6;74.7]	74.9% [73.2;76.5]	93.8% [92.1;95.2]
	-1091 (156s)	-1442 (977s)	-1483 (982s)	-50969 (22562s)	-16197 (963s)
GS+T-ACA	58.5% [46.3;69.6]	77.9% [66.7;86.2]	93.1% [87.8;96.2]	77.1% [75.5;78.6]	77.4% [74.6;79.9]
	-826 (16s)	-1052 (603s)	-1233 (52s)	-20469 (5050s)	-15391 (856s)
SEM+T	64.6% [52.5;75.1]	51.5% [39.8;62.9]	93.1% [87.8;96.2]	74.9% [73.2;76.5]	93.8% [92.1;95.2]
	-1112 (341s)	-1447 (2190s)	-1485 (1094s)	-50969 (30417s)	-15729 (5492s)
GES-ACA	64.6% [52.5;75.1]	82.4% [71.6;89.6]	93.8% [88.6;96.7]	77.1% [75.5;78.6]	96.1% [94.7;97.1]
	-866 (76s)	-1160 (536s)	-1293 (123s)	-23462 (6350s)	-15535 (515s)
GES-EM	64.6% [52.5;75.1]	51.5% [39.8;62.9]	68.3% [60.3;75.3]	74.9% [73.2;76.5]	93.8% [92.1;95.2]
	-1101 (240s)	-1446 (1120s)	-1522 (1062s)	-38947 (54748s)	-16197 (1545s)

Table 5: *Two first lines* : names of datasets; number of attributs; length of the learning dataset; length of the test dataset; percentage of incomplete entries. *Following lines* : name of method; best good classification percentage on three runs; 95%-confidence interval; selected model likelyhood; learning time in seconds on a laptop 2.4GHz with Matlab®R2006a.
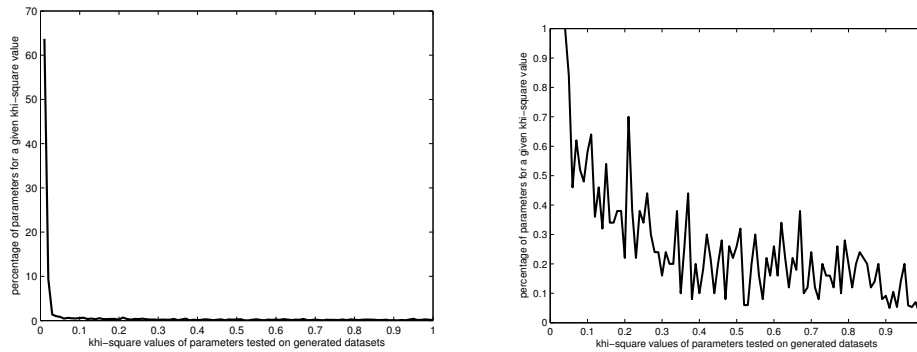

Figure 2: Histogram of χ^2 value of parameters tested from generated samples and Zoom of the flat part on the right.

Finally, the table 5 illustrates that TAN-EM and MWST-EM have about the same complexity (regarding the computational time) and are a good compromise between NB and greedy searches in DAG andCPDAG spaces.

4.5 Generating incomplete datasets

For the experimentation stage, we have used our formalism to generate datasets from randomly generated Bayesian networks (between 4 and 13 nodes). Those networks have been used to gener MAR incomplete datasets with 10000 samples with a percentage of missingness which is randomly chosen between 15% and 40% (results on MCAR datasets are similar). Then we pick up different parameters which model the percentage of missingness of an attribute in a specific context for each incomplete dataset generative Bayesian network. We then calculate the χ^2 critical value that this parameter has if we test it on the corresponding generated dataset.

In figure 2, an histogram of Chi-square values of parameters tested on generated datasets is shown.

As we could see on figure 2, the distribution of Chi-square values is high for small values (*i.e.* < 0.05) and arround 65% of the parameters tested have a Chi-square value smaller than 0.01.

On figure 2, we could see that arround 0.02% of tested parameters could have a fixed Chi-square value higher than 0.3. Those values are reach for parameters that lead to a small number of samples in the datasets. Then the tests are not reliable in this case as the number of corresponding samples is often smaller than 20 samples.

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 relations between attributes is difficult when the sample size is too small. For most methods, random initializations can be replaced effectively by initializations issued from a simple algorithm like MWST.

Our second experiment permitted to evaluate the effectiveness of these methods for classification tasks. Here, we have shown that a good structure search can lead to results similar to the k-NN method but can also be used in other ways (structure interpretating, inference on other nodes and dealing 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 parsing the Markov equivalent space (cf definition 3) instead of the DAGs space leads to optimal results. Munteanu *et al.* Munteanu and Bendou (2002) proved that this space has better properties and Chickering (2002a); Castelo and Kocka (2002) propose a new structure learning in this space. Moreover Chickering (2002a) proved the optimality of his GES method. In our experiments, this method has returned the best results regarding the scoring function, but if we consider the editing distance or the classification rate, the results are not so satisfying.

The EM algorithm, which is very often used, don't seems to give good results regarding those that are obtained using pairwise deletion. The two estimation technics leads to different results. When we want to interpret the structure, we should use EM, and when we want to perform classification or simulation task, we should prefer using ACA.

Adapting existing methods to deal with missing data is very important while dealing with realistic problems. The SEM algorithm performs 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 initialization problems are also yet to be solved. Finally, the final step could consist in adapting the Structural EM principle to Markov equivalent search methods.

In future work, we should study sensitivity of all structure learning algorithms to the fitting function that is optimised. In this study, we have used the BIC criterion, but we could also use MDL or BDe criterion for instances.

Bibliographie

References

S.K. Andersen, K.G. Olesen, F.V. Jensen, and F. Jensen. Hugin - a shell for building bayesian belief universes for expert systems. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 1:1080–1085, 1989. http://www.hugin.com/.

V. Auvray and L. Wehenkel. On the construction of the inclusion boundary neighbourhood for markov equivalence classes of bayesian network structures. In Adnan Darwiche and Nir Friedman, editors, *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 26–35, S.F., Cal., 2002. Morgan Kaufmann Publishers.

Y. Bennani and F. Bossaert. Predictive neural networks for traffic disturbance detection in the telephone network. In *Proceedings of IMACS-CESA'96*, Lille, France, 1996.

C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

H. Borchani, N. Ben Amor, and K. Mellouli. Learning bayesian network equivalence classes from incomplete data. *Lecture Notes in Computer Science*, 4265/2006:291–295, 2006.

R. R. Bouckaert. Probabilistic network construction using the minimum description length principle. *Lecture Notes in Computer Science*, 747:41–48, 1993.

G. Bradski. Open source probabilistic network library. Systems Technology Labs, Intel, 2004. http://www.intel.com/research/mrl/pnl/.

R. Castelo and T. Kocka. Towards an inclusion driven learning of bayesian networks. Technical Report UU-CS-2002-05, Institute of information and computing sciences, University of Utrecht, 2002.

J. Cheng, C. Hatzis, H. Hayashi, N.A. Krogel, S. Morishita, D. Page, and J Sese. Kdd cup 2001 repport. the awards ceremony of the 7th ACM SIGKDD 2001, pp 47-64 (San Francisco, CA), 2001. http://www.cs.ualberta.ca/~jcheng/bnsoft.htm.

J. Cheng, R. Greiner, J. Kelly, D. Bell, and W. Liu. Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, 137(1-2):43–90, 2002.

D. Chickering. Learning equivalence classes of bayesian network structures. In Eric Horvitz and Finn Jensen, editors, *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 150–157, San Francisco, August 1996. Morgan Kaufmann Publishers. ISBN 1-55860-412-X.

D.M. Chickering. Learning equivalence classes of bayesian-network structures. *Journal of machine learning research*, 2:445–498, 2002a.

D.M. Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, November 2002b. URL http://www.ai.mit.edu/projects/jmlr/papers/volume2/chickering02a/chickering02b.pdf.

C.K. Chow and C.N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

G. Cooper and E. Hersovits. A bayesian method for the induction of probabilistic networks from data. *Maching Learning*, 9:309–347, 1992.

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B 39:1–38, 1977.

D. Dor and M. Tarsi. A simple algorithm to construct a consistent extension of a partially oriented graph. Technical Report R-185, Cognitive Systems Laboratory, UCLA Computer Science Department, 1992.

N. Drakos and R. Moore. Javabayes, bayesian networks in java : User manual and download, 1998. Available at URL: http://www-2.cs.cmu.edu/~javabayes/Home/.

O.C.H. François and P. Leray. Learning the tree augmented naive bayes classifier from incomplete datasets. In *Proceedings of the Third European Workshop on Probabilistic Graphical Models (PGM'06), ISBN: 80-86742-14-8*, pages 91–98, Prague, Czech Republic, september 2006.

O.C.H. François and P. Leray. Incomplete datasets generation using the bayesian networks formalism. In *Proceedings of the International Joint Conferences on Neural Networks (IJCNN 2007)*, Orlando, Florida, 2007.

Olivier François. *De l'identification de structure de réseaux bayésiens à la reconnaissance de formes à partir d'informations complètes ou incomplètes*. PhD thesis, Institut National des Sciences Appliquées de Rouen (INSA), `http://ofrancois.tuxfamily.org/these.html`, 2006.

N. Friedman. The bayesian structural EM algorithm. In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 129–138, San Francisco, July 1998. Morgan Kaufmann. ISBN 1-55860-555-X.

N. Friedman and G. Elidan. LibB for windows/linux programs, 1999. URL `http://www.cs.huji.ac.il/labs/compbio/LibB/`.

N. Friedman, D. Geiger, and M. Goldszmidt. bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997a.

N. Friedman, M. Goldszmidt, D. Heckerman, and S. Russell. Challenge: What is the impact of bayesian networks on learning?, *Proceedings of the 15'th International Joint Conference on Artificial Intelligence (NIL-97)*, 10-15, 1997b. `http://www.cs.huji.ac.il/labs/compbio/Repository/`.

D. Geiger. An entropy-based learning algorithm of bayesian conditional trees. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighth Conference (UAI-1992)*, pages 92–97, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

D. Heckerman, D. Geiger, and M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. In Ramon Lopez de Mantaras and David Poole, editors, *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 293–301, San Francisco, CA, USA, July 1994. Morgan Kaufmann Publishers. ISBN 1-55860-3328.

F.V. Jensen. *An introduction to Bayesian Networks*. Taylor and Francis, London, United Kingdom, 1996.

M. I Jordan. *Learning in Graphical Models*. Kluwer Academic Publishers, The Netherlands, 1998.

C.M. Kadie, D. Hovel, and E. Horvitz. Msbnx: A component-centric toolkit for modeling and inference with bayesian networks. Microsoft Research Technical Report MSR-TR-2001-67, July 2001. `http://www.research.microsoft.com/adapt/MSBNx/`.

E. Keogh and M. Pazzani. 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*, pages 225–230, 1999.

J.H. Kim and J. Pearl. Convice; a conversational inference consolidation engine. *IEEE Trans. on Systems, Man and Cybernetics*, 17:120–132, 1987.

S. L. Lauritzen, J. H. Badsberg, S. G. BÃÿttcher, P. Dalgaard, C. Dethlefsen, D. Edwards, P. S. Eriksen, A. R. Gregersen, S. HÃÿjsgaard, and S. Kreiner. gr - graphical models in r. Aalborg, DENMARK: Aalborg University Department of Mathematical Sciences, 2004. Available from URL: http://www.math.auc.dk/gr/.

S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society B*, 50(2):157–224, 1988.

P. Leray and O. François. Bayesian Network Structural Learning and Incomplete Data. In *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR 2005), Espoo, Finland*, pages 33–40, 2005.

P. Leray, S. Guilmineau, G. Noizet, O. François, E. Feasson, and B. Minoc. French BNT site, 2003. http://bnt.insa-rouen.fr/.

E. Mazer, J-F. Miribel, P. Bessière, O. Lebeltel, K. Mekhnacha, and J-M. Ahuactzin. Probayes : Mastering uncertainty, 2004. http://www.probayes.com/.

S. Meganck, P. Leray, and B. Manderick. Learning causal bayesian networks from observations and experiments: A decision theoretic approach. In *Modelling Decisions in Artificial Intelligence (MDAI'06)*, 2006.

D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning*, *Neural and Statistical Classification*, 1994. http://www.amsta.leeds.ac.uk/~charles/statlog/, http://www.liacc.up.pt/ML/statlog/datasets/.

P. Munteanu and M. Bendou. The eq framework for learning equivalence classes of bayesian networks. In *First IEEE International Conference on Data Mining (IEEE ICDM)*, pages 417–424, San José, November 2002.

P. Munteanu, L. Jouffe, and P.H. Wuillemin. Bayesia lab, 2001. URL http://www.bayesia.com/. http://www.bayesia.com/.

K. Murphy. Active learning of causal bayes net structure. Technical report, UC Berkeley, 2001.

K.P. Murphy. Bayes net toolbox v5 for matlab. Cambridge, MA: MIT Computer Science and Artificial Intelligence Laboratory, 2004. http://bnt.sourceforge.net.

P. Myllymäki, T. Silander, H. Tirri, and P. Uronen. B-course: A web-based tool for bayesian and causal data analysis. International Journal on Artificial Intelligence Tools, **11**(3), p. 369-387, 2002. http://b-course.hiit.fi/.

Netica. by Norsys Software Corp., 1998. http://www.norsys.com/.

M. Nijman, E. Akay, W. Wiegerinck, and S.N.N. Nijmegen. Bayesbuilder: A tool for constructing and testing bayesian networks, 2002. Available at URL: http://www.snn.kun.nl/nijmegen/index.php3?page=31.

P. Courtellement O.Colot, C. Olivier and A. El Matouat. Information criteria and abrupt changes in probability laws. *Signal Processing VII : Théorie and Applications*, pages 1855–1858, 1994.

J. Pearl and T.S. Verma. A theory of inferred causation. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *KR'91: Principles of Knowledge Representation and Reasoning*, pages 441–452, San Mateo, California, 1991. Morgan Kaufmann.

B. P. Perry and J. A. Stilson. Bn-tools: A software toolkit for experimentation in bbns (student abstract). In Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-2002), Edmondon, Alberta, CANADA, pp. 963-964. Menlo Park, CA: AAAI Press, 2002. Avalaible at URL: http://bnj.sourceforge.net/.

R. W. Robinson. Counting unlabeled acyclic digraphs. In C. H. C. Little, editor, *Combinatorial Mathematics V*, volume 622 of *Lecture Notes in Mathematics*, pages 28–43, Berlin, 1977. Springer.

D.B. Rubin. Inference and missing data. *Biometrika*, 63:581–592, 1976.

G Schwartz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2): 461–464, 1978.

P. Sebastiani, M. Ramoni, and A. Crea. Profiling your customers using bayesian networks: A tutorial exercise and the bayesware. KDD Cup 99, 1999. http://bayesware.com/.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. The MIT Press, ISBN: 0-262-19440-6, 2 edition, 2000.

P. Spirtes, C. Glymour, and R. Scheines. The tetrad project: Causal models and statistical data. pittsburgh. PA: Carnegie Mellon University Department of Philosophy, 2004. Available from URL: http://www.phil.cmu.edu/projects/tetrad/.

A. I. Sutherland and R. J. Henery. Statlog - an ESPRIT projecy for the comparison of statistical and logical learning algorithms. *New Techniques and Technologies for Statistics*, February 1992.

I. Tsamardinos, L. Brown, and C. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. Technical Report DSL-TR-05-01, Vanderbuilt University, 2005. http://www.dsl-lab.org.

T. Verma and J. Pearl. Equivalence and synthesis of causal models. In *Proceedings Sixth Conference on Uncertainty and Artificial Intelligence*, pages 255–268, San Francisco, 1990. Morgan Kaufmann.

Y. Xiang. WebWeavR-III, 1999. http://snowhite.cis.uoguelph.ca/faculty_info/yxiang/ww3/.