Incremental Methods for Bayesian Network Learning

Josep Roure Ramon Sangüesa

September 20, 1999

Abstract

In this work we analyze the most relevant, in our opinion, algorithms for learning Bayesian Networks. We analyze methods that use goodness-of-fit tests between tentative networks and data. Within this sort of learning algorithms we distinguish batch and incremental methods. Finally, we propose a system, called BANDOLER, that incrementally learns Bayesian Networks from data and prior knowledge. The incremental fashion of the system allows to modify the learning strategy and to introduce new prior knowledge during the learning process in the light of the already learnt structure.

1 Introduction

The aim of this work is twofold. On the one hand, we introduce the state of the art on learning Bayesian networks. It is intended to be a tutorial on the learning methods based on goodness-of-fit tests. We present the most significant, in our opinion, learning algorithms found in the literature, as well as the theory they are based on. On the other hand, we propose a research framework. The field of learning Bayesian networks has been largely developed from the numerical (i.e. statistical) viewpoint. As we shall see, we belief that it is worth developing this field from the Machine Learning viewpoint.

Here is a brief sketch of what the remaining sections deal with:

Machine Learning We recall some concepts and definitions from the Machine Learning community. Our aim is to state some basic concepts, which we will use to analyze the algorithms proposed by the Bayesian network learning community. We also will base our research framework proposal on these concepts.

More precisely, we introduce Machine Learning as a search process and present unsupervised learning. We argue that, in the context of unsupervised learning, background knowledge may be very important since it is a source of bias for any learning process. We also introduce the incremental learning approach and learning when the world changes over time.

Bayesian Networks We briefly recall the definition of the Bayesian networks and the concept of d-separation.

Learning Bayesian Networks: batch methods We go through the most significant Bayesian network learning algorithms. We review the methods based on goodness-of-fit tests between the probability distribution of a tentative structure and the true joint distribution implied by data.

We divide methods accordingly to the sort of test they use. Namely, tests based on information theory, Bayesian theory and Minimum Description Length approach. We introduce the basic concepts on each of these sort of test.

Learning Bayesian Networks: the incremental approach We consider the main bibliography about incremental learning algorithms for Bayesian networks. Up to 1998, there are only three proposals for incremental algorithms. The aim of this sort of algorithms is to modify an already learnt structure when new data is available.

At the end of the section we compare the reviewed algorithms with the definition given in the **Machine Learning** section. We also compare the the three proposals between them.

Our Proposal We propose a framework for research on learning Bayesian networks. The framework recalls some of the concepts proposed in the Machine Learning field and applies them to learning Bayesian networks. The proposed framework is quite comprehensive, hence we choose some parts of it to be developed in our PhD thesis. We belief that the rest of the framework is a research field worth being developed in the future.

2 Machine Learning

Artificial Intelligence has proposed several models in order to describe domains. Most of these models have successfully been used for diagnosis, prediction and decision making. Real-world applications made evident the need for automatic knowledge elicitation processes in order to construct such domain models. The need appeared for two reasons, firstly the elicitation of knowledge from experts has always been a difficult and time-consuming task and, secondly, large databases are becoming increasingly abundant in many areas like science, engineering and business, so it is natural to explore them in order to derive new knowledge.

Knowledge acquisition from experts is difficult because of different well known reasons. Sometimes communication between domain experts and knowledge engineers is difficult as they come from different knowledge areas and therefore speak different jargons. Other times there are few, if any, experts on a given domain and it is almost impossible to reach them. Also, when domains are complex or ill-structured it is simply impossible for experts to give a precise model [2]. However, it is not always expert's *shortcomings* what cause the difficulties of the knowledge elicitation process. Many times, domain models may be very large and may need large collections of precise quantitative information (i.e. real numbers). Computers are much better and faster in processing large collections of data than human beings. When large databases are available, automatic elicitation process from data are needed because processing data manually would take an unfeasible amount of time. Databases may be a rich source of knowledge of great interest. Knowledge is "hidden" among the vast amounts of data and must be *distilled* into domain models. The *distillation* processes are called *learning* algorithms by the artificial intelligence community.

2.1 Machine Learning as search

We may begin with a broad and general definition of Machine learning:

> **Definition:** A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

This definition, stated by Tom. M. Mitchell [25], includes any computer system that improves its performance at tasks through experience. As we shall see, we are interested in a particular sort of Machine Learning.

When the system improves its performance at tasks by means of rising the quality of its model of the domain, and, in turn, the domain model is learnt from the information in a database, we say that the system performs *inductive inference learning.*

Michalski and Ram [23] characterize inference learning as a process of *inferential search* through a knowledge space. The knowledge space is a space of knowledge representations that can represent all possible inputs, all of the learner's background knowledge, and all knowledge that the learner can potentially generate. The search is done by the application of *knowledge transmutations.* Knowledge transmutations are operators that make changes to the knowledge representations in the knowledge space. In order to characterize the inferential search they consider the following entailment

$$P \cup BK \supset C$$

where P stands for a set of statements called the premise, BK stands for a set of statements representing the reasoner's background knowledge, and C stands for a set of statements called the consequent. P is assumed to be consistent with BK. *Inductive* learning hypothesizes (learns) the premise P, given C and BK. On the contrary, *deductive* learning derives the consequent C given P and BK.

We, in the inductive inference approach, shall think of C as the database (the description of a set of consequences from the domain) and of P as the domain the system learns. Note that actually the system learns a model P' of the domain P.

Michalschi and Ram's definition casts learning as a searching process among the knowledge space. The system, in order to perform this search, needs the following components:

- A **language** to describe the domain models. This description may be partial when the system has not yet gained a full picture of the domain. These descriptions, including partial ones, are usually said to form the knowledge or search space.
- A **set of operators** in order to perform the knowledge transmutations. The learning system uses these operators to transform an already existing domain description into a new, possibly better, one. The system moves through the search space, using these operators, looking for the best model.
- An **evaluation function** that measures the quality of the domain descriptions. This function is used in order to compare different domain models (or knowledge states).
- A **search strategy** in order to find the best model description. Usually the search space is so huge that it is not computationally feasible to perform an exhaustive search. Therefore, some search strategy (i.e. hill-climbing) or some heuristics is needed in order not to search all the knowledge space. Usually strategies are not able to reach the *best* domain model but they may find a reasonably good one.

Finally, we may distinguish between *supervised* and *unsupervised* learning. In the former, there is a teacher that knows in advance the structure of the domain. The teacher supplies significant examples (or consequences) together with the structure of the domain to the learning system. Thus, the learning system only has to build a description of the domain structure by inducing it from the examples. On the other hand, the *unsupervised* learning system has to extract both the structure and its description from a dataset hoping that they are a representative sample of the domain.

Our work shall focus on inductive unsupervised learning, since our aim is to study the field of Bayesian network learning. As we shall see, in later sections, learning Bayesian networks may be classified as a sort of inductive, unsupervised and data-driven learning.

2.2 Background knowledge and the role of bias in Machine Learning

At the beginning of this section we have already argued that sometimes there is not a precise model of a domain when it is complex or ill-structured. Other times it is difficult to find an expert able to supply both the domain structure and examples in order to perform inference and gain a computerized domain description.

However, even an expert cannot give a precise and complete domain description, she may be able to provide a partial one. This partial description given by an expert is usually called *background* or *prior* knowledge. This sort of knowledge may help in two ways. First, the knowledge may narrow the search space, by discarding solutions which may be bad, and let the search process to rapidly form on a good description. Second, the knowledge may be used to guide the search towards that part of the knowledge space where best domain descriptions are obtaining, hence, solutions of higher quality.

It is also widely accepted in the literature that biases play an important role in Machine Learning. For example Tom M. Mitchell [24] states that although removing all biases from a learning system may seem to be a desirable goal, in fact the result is nearly useless.

Before going ahead, we should give a definition for bias. Following Tom M. Mitchell [24] and Gordon and Desjardins [14] we may say

> **Definition:** Bias is any basis for choosing one domain model over another, other that strict consistency with the instances.

It is not clear in this definition whether background knowledge should be considered a kind of bias or, on the contrary, they are different concepts. We agree with Gordon and Desjardins [14] who say that background knowledge has the supportive role of providing information to select a bias. Hence, it cannot be considered to be a bias *per se*.

Gordon and Desjardins also argue that biases should not be wired or embedded within the search heuristics. Biases should be kept separated from the heuristics and clearly identifiable in the program. In this way the programmers will be able to easily modify biases in case they are faulty. Gordon and Desjardins also argue that it could be desirable to have *dynamic* biases in order to automaticly adapt biases to the specific learning problem during the process. See [14] for a deeper discussion.

There are many types of biases that can be applied in order to improve the performance of the learning processes. We may classify biases depending on which of the different components of a search learning process is used:

- Database: biases may be applied in order to *filter* the examples of the database. For example when it is known that certain variable does not provide any useful information or when it is known that some kind of examples are not significant, they should not be used to learn the domain model.

- Language to describe the domain models: it may be possible to limit the representational power of the language. It could be desirable that the language cannot represent those models that are known beforehand to be of low quality.

- Set of operators: we may be able to construct operators that are known to perform useful knowledge transmutations.

- Evaluation function: evaluation function may score more factors than merely accuracy with the database. For example, the function may score higher those domain models that are simpler in front of those which are a bit more accurate but much more complex.

- Search strategy: biases may be used to guide the search through the knowledge space. For example in a beam search the bias may indicate how many beams should be explored. Also, biases may be used to choose the transmutation operator to be applied.

So, we have seen that there are many different kinds of biases that can be used to achieve rapidly a good domain model or to gain a more accurate one. We think that it is worth keeping biases at another level, separated from the learning algorithm itself.

2.3 Machine Learning: The Incremental approach

Machine learning algorithms may also be classified according to the way they process the data. Learning algorithms may be batch or incremental. Batch algorithms, given the whole training data set, output a domain model after processing, possibly multiple times, data. This sort of algorithms stop learning when they have processed the dataset and assume that they have reach a good domain model which will be used for the whole *life* of the artificial agent. On the other hand, incremental algorithms never assume they reach a final learning stage. They keep improving their domain model by processing new data items as long as they are available. Incremental algorithms process each single item of data as it is available without reprocessing previously seen ones. In this way, during the whole learning process there is a domain model available, although incomplete, that can be used for whatever task it is intended.

Incremental learning algorithms have desirable properties that let them overcome some problems found in real-world application. These problems could be resumed as:

1. Time limitation: when databases are so huge that must be stored on secondary memory, multiple inspection of such amount of data is unfeasible. Incremental methods should process data items only once.

2. Any-time availability: sometimes, given the nature of the real-world application, an *intelligent* agent needs to use a domain model in order to carry out its performance task even if the whole dataset is not available. Incremental methods can deal with such situations because they have a domain model during the whole learning process.

3. Changing worlds: when intelligent agents must *survive* in a changing world they should be able to make their model of the world evolve. Incremental algorithms are a natural solution to cope with such situations because they are able to incorporate into the model new samples from the changing world.

2.3.1 Incremental algorithms: a definition

Incremental learning algorithms have been largely studied in Conceptual Clustering [1, 11, 22] and Concept Formation [13, 2] communities.

The idea of incrementality arises from the observation that much of human learning can be viewed as a gradual process of concept formation and human ability for incorporating knowledge from new experiences into already learnt concept structures. The incremental learning approach is firstly motivated, in the Concept Formation community, as a human capability worth being held by artificial agents.

In the Bayesian network community, incremental learning algorithms are an area of recent and growing interest [4, 20, 12]. The fact that Bayesian network community has a strong numerical (statistical) background may explain its late interest in incremental learning. Conceptual Clustering and Concept formation share some characteristics with learning Bayesian networks. Namely, both perform induction from a dataset and hence are data-driven. Also both fall into the unsupervised learning category, that is, algorithms are not provided neither with a domain structure nor with selected examples by a teacher.

All these facts motivated us to begin our study of incremental learning algorithms in Conceptual Clustering and Concept Formation areas. In these areas databases are called *training data sets* as they are used to *train* software agents, and by a *training experience* it is meant a single training instance or a single sample from a database. Langley [21] defined precisely what an incremental algorithm is.

> **Definition:** A learner L is incremental if L inputs one training experience at a time, does not reprocess any previous experiences, and retains only one knowledge structure in memory.

Langley's definition is rather strong, because it imposes three constraints to an algorithm in order to be incremental. The first two constraints require learning agents to be able to use their knowledge at any time of the learning process. More precisely, his second constraint rules out those agents that process new data together with old one in order to come out with a new model. The important idea of this constraint is maintaining reasonably low and constant the time required to process each data instance over all the data set. The third constraint tries that learning agents do not make unreasonable memory demands.

We also want to remark that a learning algorithm may not be incremental because it learns samples one by one but because it learns variables incrementally as they are available [21]. If we see a database as a matrix where rows are samples and columns are variables describing samples, an algorithm could incrementally learn variables (columns) instead of samples (rows). In this way, an incremental algorithm grows a domain model incorporating variables to it as they are available. We shall see that some of the learning algorithms presented in the Bayesian network literature as batch methods could actually be considered incremental since they learn variables in an incremental way (i.e. Herskovits and Cooper's K2 algorithm [8]).

2.3.2 Incremental algorithms: search strategies and their drawbacks

Most incremental learning algorithms use one of two search algorithms, namely hill climbing or beam search. Hill climbing is a search method in which one applies all operator instantiations, compares the resulting domain models using a quality evaluation function, selects the best model, and iterates until no more progress can be made. The beam search strategy can be seen as a search with several searching streams, each being a hill-climbing. The main advantage of hill climbing is its low time and memory requirements since there are never more than a few search states (or domain models) in memory and thus searching paths (or streams) to be explored.

However, hill climbing also suffers from well-known drawbacks, such as its tendency to halt at local optima and a dependence on step size. In order to overcome these problems beam search is used to maintain different searches and, in this way, explore different optima. However, the algorithms using the beam search are usually provided with parameters which state the number of beams to be used in order to control the amount of time and memory required to perform the search.

In addition, it is observed [10, 13, 21, 22, 2] that incremental hill climbing, in the context of unsupervised learning, is order sensitive. Namely, given two sample orders, O_1 and O_2, of a database D an incremental hill climbing algorithm may output different domain models when fed with order O_1 or with order O_2. Ordering effects are due to the nature of incremental process of data combined with the tendency of hill climbing methods to stick to local maxima. This sort of algorithms may output a very skewed model when the firstly observed samples give a biased view of the domain even if the last samples give a correct one [10]. Stating this problem in another way, it may happen that firstly seen data guide the learning process to a local maxima surrounded by *deep valleys*. Therefore, when new data is available it is very difficult for the hill climbing strategy to fly off the deep valleys. The capability of flying off valleys depends very much on the step size of the knowledge transmutation operators. Local minima and ordering problems may also be found in beam search algorithms but at a minor scale. The more beams are maintained, the less the search is affected from order and local minima problems.

Research in incremental clustering has approached the ordering effects problem by using several strategies. Roure and Talavera [31, 33] reported a classification of strategies to avoid ordering effects in incremental clustering algorithms. They also proposed an extension of a strategy firstly proposed by Béjar [2] and Lebowitz [22]. Roughly speaking it consists on deferring the incorporation of those samples that seem to guide, at that moment of the learning process, the search to a local maxima.

We would like to remark, that the methods proposed in order to overcome the problems of incremental algorithms relax in some way the three hard constrains of Langley's definition. These allow agents to input more than one instance at a time, allow limited reprocessing of data and also allow keeping in memory few alternative domain models. Anyway, all these methods try to follow the ideas which are behind Langley's constraints.

3 Bayesian Networks

Bayesian networks are graphical representations of causal relations between variables in a domain. Bayesian networks are also called belief networks, Bayesian belief networks or causal probabilistic networks and they use probability theory in order to reason with uncertainty.

The advantage of graphical representations is that they allow people to express directly the fundamental qualitative relationship of direct causation. The arcs between variables signify the existence of direct causal influences and the strengths of these influences are quantified by conditional probabilities.

3.1 Probabilistic networks

Probability theory views a domain U as a set of random variables $\mathbf{U} = \{X_1, \ldots, X_n\}$ each of which has a domain of possible values. The key concept in probability theory is the *joint probability distribution*, which specifies a probability for each possible combination of values for all the random variables. Given this distribution, one can compute any desired posterior probability given any combination of evidence.

Unfortunately, an explicit description of the joint distribution requires a number of parameters that is exponential in n, the number of variables. Probabilistic networks derive their power from the ability to represent conditional independences among variables, which allows them to take advantage of the *locality* of causal influences.

A *Bayesian network* is an annotated directed acyclic graph that encodes a joint probability distribution of a set of random variables $\mathbf{U}$. Formally, a Bayesian network for $\mathbf{U}$ is a pair $S = (B_S, B_P)$:

- The first component, B_S, is a directed acyclic graph (DAG) whose vertices corresponds to the random variables $X_1, \ldots, X_n$, and whose edges represent directed dependencies between variables.

  Let us give a more detailed explanation. We say that two sets of variables $\mathbf{X}$ and $\mathbf{Y}$ are independent given $\mathbf{Z}$ if $P(X|Y,Z) = P(X|Z)$ whenever $P(X,Y) > 0$. Recall the chain rule of probability,

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | X_1, \ldots, X_{i-1})$$

If for each variable X_i, the set $\mathbf{Pa_i} \subseteq \{X_1, \ldots, X_{i-1}\}$ renders X_i and $\{X_1, \ldots, X_{i-1}\}$ independent, that is, $P(X_i|X_1, \ldots, X_{i-1}) = P(X_i|\mathbf{Pa}_i)$ then one can rewrite the chain rule as

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | \mathbf{Pa}_i) \tag{1}$$

A Bayesian network structure B_S encodes the assertion of conditional independence in equation 1. Namely, B_S is a DAG where each variable in $\mathbf{U}$ corresponds to a node in B_S, and the parents of the node corresponding to X_i are the nodes corresponding to the variables in $\mathbf{Pa}_i$.

- The second component, B_P, represents the set of parameters that quantifies the network. It contains a parameter $\theta_{ijk} = P(X_i = x_i^k | \mathbf{Pa}_i = \mathbf{pa}_i^j)$ for each possible state x_i^j of X_i and for each configuration $\mathbf{pa}_i^j$ of $\mathbf{Pa}_i$

3.2 D-separation

As we have said above, Bayesian networks represent direct causation between variables. The power of Bayesian networks is that they have built-in independence assumptions which are not explicitly specified.

The independence assumptions are read from a network structure by means of the *d-separation* criterion. In order to understand the d-separation we need to keep in mind the three basic connections between variables:

1. **Serial connection**: Consider the situation in Figure 1(a). A has an influence on B which in turn has influence on C. Obviously, evidence on A will influence the certainty of B which then influences the certainty of C. Similarly, evidence on C will influence the certainty of A through B. On the other hand, if the state of B is known, then the channel is blocked, and A and C become independent. We say that A and C are d-separated given B.

2. **Diverging connection**: In the situation in Figure 1(b) the influence can pass between all the children of A unless the state of A is known. We say that $B, C, \ldots, X$ are d-separated given A.

3. **Converging connections**: In this situation, Figure 1(c), if nothing is known about A except what may be inferred from knowledge of its parents $B, C, \ldots, X$, then the parents are independent, that is, evidence on one of them has no influence on the certainty of the others. Now, if any other kind of evidence influences the certainty of A, then the parents become dependent due to the principle of explaining away. The evidence may be direct evidence on A, or it may be evidence from a child.

The three cases above cover all the ways in which evidence may be transmitted through a variable, and following the rules it is possible to decide for any pair of variables in a causal network whether they are dependent given the evidence entered into the network.

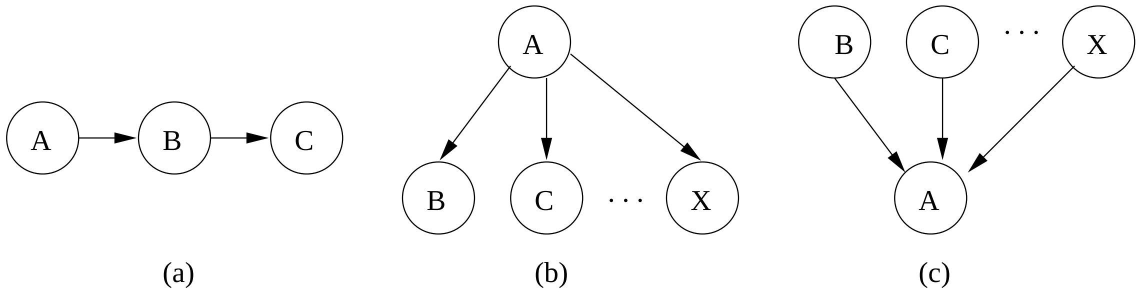

Figure 1: The serial, converging and diverging connections

Definition: Two variables A and B in a causal network are **d-separated** if for all paths between A and B there is an intermediate variable V such that either the connection is serial or diverging and the state of V is known; or the connection is converging and neither V nor any of V's descendants have received evidence.

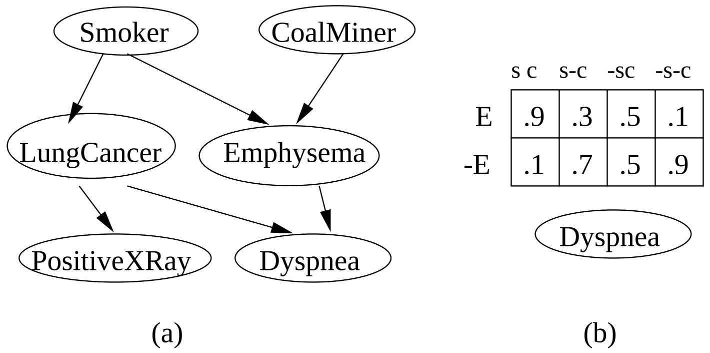

Figure 2: A Bayesian network example

In Figure 2 we have an example of a Bayesian Network. On the left side (a), we have the network structure B_S representing the independencies among the variables of the domain. On the right side (b), we have the probability table θ_{emph} for the variable *Emphysema*

Note that in this network, if we have some evidence on *Dyspnea*, *Smoker* and *CoalMiner* are d-connected, otherwise they are d-separated. If we know the state of *Lungcancer*, *Dyspnea* and *PositiveXRay* are d-separated. Also *Somker* and *Dyspnea* are d-separated when the state of *Lungcancer* is known.

For further reading in general theory on Bayesian networks the reader is referred to the Pearl (1988) [29], Neapolitan (1990) [26], Jensen (1998) [18] and a short introduction Charniak (1991) [6].

4 Learning Bayesian Networks: batch methods

The Bayesian network learning algorithms aim at finding the network, or a reduced set of networks, that best encodes the joint probability distribution of data. The learning problem of this kind of networks can be stated as follows [32]:

Given a dataset, infer the topology for the belief network that may have generated the dataset together with the corresponding uncertainty distribution.

This process of inferring a Bayesian network from data can be seen as search process looking for the best network. Thus, Bayesian network learning algorithms fall indeed into the category of unsupervised (neither a network non a selected set of examples are given by a teacher), inductive inference (given a set of consequences, that is a dataset, the algorithm hypothesizes a domain model), and data-driven (data guides the searching process).

In general, one can distinguish two great groups of methods. The first ones are based on the application of conditional independence tests between variables and the construction of the structure of the network based on the result of such tests; then the conditional probability tables are calculated from data. The second ones are methods based on goodness-of-fit tests between the probability distribution of a tentative structure and the true joint distribution implied by data.

We will focus our work on this second category of methods. The main drawback of the methods based on conditional independence information is that they need a source providing independency statements. Independency statements can be derived from data using statistical tests. However, when there is a weak dependence between two variables, or when binary variables are involved, those tests require large databases to return reliable results. Together with the restriction that the independency statements represented by the network structure are exactly those in the domain, these methods are in general impractical for small databases with discrete variables [3]. For us, this is an unbearable problem as we want to develop incremental algorithms able to evolve the Bayesian networks structures when new, possibly few, data items are available.

Within the category of methods based on goodness-of-fit tests we can distinguish different approaches to such tests or quality measures. Namely, Cross Entropy [7], Bayesian inference [8, 15] and Minimum Description Length (MDL) [19]. All these approaches have derived some form of establishing the overall quality of a network in terms of its constituents, reducing quality measures to the sum of the quality of all given child-parent configurations. This is possible thanks to the property of factorization over distribution which is inherent to Bayesian networks:

$$\text{Quality}(\text{Network}|\text{Dataset}) = \sum_{X_i} \text{quality}(X_i|\mathbf{Pa}_i, \text{Dataset})$$

where $\mathbf{Pa}_i$ is the set of parents of variable X_i.

During the rest of this section we briefly review the theory underlying each approach (Entropy, Bayesian inference and MDL) followed by a review of the most significant learning algorithms. We review the algorithms in strict temporal order which, curiously, coincides with the three approaches as listed above.

We begin our revision with the algorithm proposed by Chow and Liu (1968) [7]. They are considered to have developed the first method for constructing network structures (i.e. trees), in a moment when Bayesian networks were still to be defined. They proposed a measure, based on Cross-entropy, able to calculate the quality of the whole structure by considering the local properties (i.e. factorization) of it.

Another algorithm, called Kutató, based on the Entropy approach, was developed by Herskovitz and Cooper (1990) [17]. The main contribution of this work was the algorithm itself. By taking the idea of the local structure of networks, they proposed an heuristic in order to find a Bayesian network (i.e. DAG) of high quality. After this work, Cooper and Herskovitz (1992) [8], proposed another algorithm, called K2. The K2 algorithm, in fact, is the Kutató algorithm using a new quality measure. The new quality measure falls in the

Bayesian inference category, giving statistical soundness to the learning of Bayesian networks field.

And last, we review the algorithm proposed by Lam and Bacchus (1994) [19]. They observed that the methods above prefer more accurate networks, even if their structure is much more complex. It is well known that networks with high connectivity are computationally, both time and space, more demanding and in addition they are conceptually more complex. Thus, they introduced a new quality measure, based on the Minimum Description Length approach, that performs a tradeoff between accuracy and complexity of the learnt Bayesian network. The algorithm itself is also a contribution as it uses different heuristics than the previous ones.

Despite these different approaches to quality measures seem to be very different, Bouckaert [3] demonstrated that the asymptotic behavior of the three approaches for databases of infinite size is the same and that they will yield approximately the same results for databases where all configurations of parent sets occur at least once.

4.1 Entropy-Based Methods

Entropy is a non-negative measure of the information content of a distribution. It also can be seen as a measure of the uncertainty of a given variable. Entropy is defined as following:

$$H(X) = -\sum_{i=1}^{r} P(x_i) \log P(x_i) \tag{2}$$

Where r denotes the number of possible states for variable X. We now note some properties of the entropy function:

- $H(X) \geq 0$ with equality iff $P(x_i) = 1$ for one i.
- $H(X) \leq \log(1/r)$ with equality iff $P(x_i) = 1/r \ \forall i = 1, \ldots, r$
- $H(X) = H(Y)$ if $\forall i = 1, \ldots, r \ P(x_i) = P(y_i)$ and $p(y_{r+1}) = 0$

The first property states that entropy is a positive measure and reaches its lowest value when the distribution's uncertainty is minimum. The second property states that entropy reaches its highest value when the distribution's uncertainty is maximum. And finally, the third property says that adding an impossible value to a distribution its entropy does not change.

The **joint entropy** of X, Y is:

$$H(X,Y) = -\sum_{i,j=1}^{r_X, r_Y} P(x_i, y_j) \log P(x_i, y_j) \tag{3}$$

The **conditional entropy of X given $Y = y_j$** is the entropy of the conditional distribution $P(X|Y = y_j)$

$$H(X|Y = y_j) = -\sum_{i=1}^{r_X} P(x_i|Y = y_j) \log P(x_i|Y = y_j) \tag{4}$$

The **conditional entropy of X given Y** is the average over y of the conditional entropy of X given y

$$H(X|Y) = -\sum_{j=1}^{r_Y} P(y_j) \left[\sum_{i=1}^{r_X} P(x_i|y_j) \log P(x_i|y_j) \right] =$$

$$-\sum_{i,j=1}^{r_X, r_Y} P(x_i, y_j) \log P(x_i|y_j) \tag{5}$$

This measures the average uncertainty that remains about X when Y is known.

The joint entropy and conditional entropy are related by the **chain rule**:

$$H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y) \tag{6}$$

The **mutual information** between X and Y measures the average reduction in uncertainty about X that results from learning the value of Y, or vice versa. Equivalently, it measures the average amount of information that Y conveys about X

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) \\ &= \sum_{i,j=1}^{r_X, r_Y} P(x_i, y_j) \log \frac{P(x_i, y_j)}{P(x_i)P(x_j)} \end{aligned} \tag{7}$$

The mutual information satisfies $I(X;Y) = I(Y;X)$ and $I(X;Y) \geq 0$

The *cross entropy* or **Kullback-Leibler divergence** between two probability distributions P and P_τ

$$D_{KL}(P||P\tau) = \sum_{i}^{r_X} P(x_i) \log \frac{P(x_i)}{P_\tau(x_i)} \tag{8}$$

The relative entropy satisfies $D_{KL}(P||P\tau) \geq 0$ (Gibbs' inequality) with equality only if $P = P_\tau$. Note that in general $D_{KL}(P||P\tau) \neq D_{KL}(P_\tau||P)$, so D_{KL} is not a distance.

4.1.1 The Chow and Liu algorithm

Chow and Liu [7] designed an algorithm to estimate the underlying n-dimensional discrete probability distribution from a set of samples. The algorithm yields as an estimation the product of $n-1$ second order distributions that optimally approximates the probability distribution. This product can also be formulated like a distribution of $n-1$ first order dependence relationships among the n variables, forming a tree dependence structure.

The algorithm uses the mutual information (7) as closeness measure where $P(\mathbf{X})$ is the probability distribution from a set of samples, and $P_\tau(\mathbf{X})$ is the tree dependence distribution. It is an optimization algorithm that gives the tree distribution closest to the distribution from the samples. Let us give some notation in order to explain the Chow and Liu's measure and algorithm.

Let $\mathbf{X} = \{X_i|1, 2, \cdots, n\}$ be a set o variables, let $j(i)$ be a mapping with $0 \leq j(i) \leq i$, let $\tau = (\mathbf{X}, E)$ be a dependence tree where $\mathbf{X}$ is the set of nodes, $E = \{(X_i, X_{j(i)})|1, 2, \cdots, n\}$ is the set of branches, and where X_0 is the *null* node. If we now assign a weight $I(X_i; X_{j(i)})$ to every dependence tree branch, the **maximum-weight** dependence tree is defined as the tree τt such that for all τ' in $\mathcal{T}_n$, $\sum_{i=1}^n I(X_i; X_{j(i)}) \geq \sum_{i=1}^n I(X_i; X_{j'(i)})$.

Chow and Liu applied some transformations to Kullback-Leibler divergence $D_{KL}(P, P_\tau)$,

$$\begin{aligned} D_{KL}(P, P_\tau) &= \sum_{\mathbf{X}} P(\mathbf{X}) \log P(\mathbf{X}) - \sum_{\mathbf{X}} P(\mathbf{X}) \sum_{i=1}^{n} \log P(X_i | X_{j(i)}) \\ &= \sum_{\mathbf{X}} P(\mathbf{X}) \log P(\mathbf{X}) - \sum_{\mathbf{X}} P(\mathbf{X}) \sum_{i=1, j(i) \neq 0}^{n} \log \frac{P(X_i, X_{j(i)})}{P(X_i) P(X_{j(i)})} \\ &\quad - \sum_{\mathbf{X}} P(\mathbf{X}) \sum_{i=1}^{n} \log P(X_i) \end{aligned}$$

Since $P(X_i)$ and $P(X_i, X_{j(i)})$ are components of $P(\mathbf{X})$,

$$-\sum_{\mathbf{X}} P(\mathbf{X}) \log P(X_i) = -\sum_{x_i} P(X_i) \log P(X_i) = H(X_i)$$

and

$$\sum_{\mathbf{X}} P(\mathbf{X}) \log \frac{P(X_i, X_{j(i)})}{P(X_i) P(X_{j(i)})} = \sum_{x_i, x_{j(i)}} P(X_i) \log \frac{P(X_i, X_{j(i)})}{P(X_i) P(X_{j(i)})} = I(X_i; X_{j(i)})$$

and obtained the following expression for the Kullback-Leibler divergence:

$$D_{KL}(P, P_\tau) = -\sum_{i=1}^{n} I(X_i; X_{j(i)}) + \sum_{i=1}^{n} H(X_i) - H(\mathbf{X}) \tag{9}$$

From the expression above, it is observed that $H(\mathbf{X})$ and $H(x_i)$ for all i are independent of the tree dependence distribution. Thus, since $D_{KL}(P, P_\tau)$ is non-negative, minimizing the closeness measure $D_{KL}(P, P_\tau)$ is equivalent to maximizing the term $\sum_{i=1}^{n} I(X_i; X_{j(i)})$. This result allowed Chow and Liu to use the Kruskal algorithm for the construction of trees of maximum total length where $I(X_i; X_{j(i)})$ may represent the distance length from node X_i to node $X_{j(i)}$. An undirected graph is formed by starting with a graph without branches and adding a branch between two nodes with the highest mutual information. Next, a branch is added which has maximal mutual information associated but does not introduce a cycle in the graph. This process is repeated until the $\frac{n(n-1)}{2}$ branches with maximum mutual information associated are added.

It is important to note that, for a given distribution, the algorithm can recover different maximal trees depending on the order in which pairs with the same weight are selected. It can be written in an algorithmical way as algorithm 1

4.1.2 The Kutató algorithm

Herskovits and Cooper [17] designed a method to learn the structure of a Bayesian network given a dataset. They recover the minimum entropy DAG, that is, the DAG whose associated joint probability distribution minimizes entropy. The network with the lowest entropy is considered to be the most informative one. Obviously, the network entropy is greater or equal to the joint entropy associated to the dataset.

Entropy for a Bayesian network structure B_S is calculated as the sum over all n variables X_i of their conditional entropy given its parents $\mathbf{Pa_i}$ in the network.

$$H(B_S) = \sum_{i=1}^{n} \left(\sum_{j}^{q_i} P(\mathbf{pa}_i^j) \sum_{k}^{r_i} P(x_i^k | \mathbf{pa}_i^j) \log P(x_i^k | \mathbf{pa}_i^j) \right) \tag{10}$$

Algorithm 1 Chow and Liu

```
Require: a database $D$ on $\mathbf{X} = \{X_1, \cdots, X_n\}$ variables
Ensure: $\tau$ be a dependence tree structure
1: $\tau = \{\emptyset\}$ the empty tree; $V = \mathbf{X}$
2: Calculate weights for every pair $I(X_i; X_j)$
3: Select the maximum cost pair $(X_i, X_j)$
4: $\tau = (X_i, X_j)$; $V = V - \{X_i, X_j\}$
5: repeat
6:    Select the maximum cost pair $(X_i, X_j)$ where $(X_i, X_k)$ or $(X_k, X_i) \in \tau$ and $X_j \in V$
7:    $\tau = \tau \cup (X_i, X_j)$, $V = V - \{X_j\}$
8: until $V = \emptyset$
```

where r_i is the number of states of variable X_i, X_i^k represents the k-th state of the variable, q_i represents the number of configurations of the parents $\mathbf{Pa_i}$, and $\mathbf{pa_i^j}$ is the j-th configuration of the parents.

The algorithm begins with a network structure with all variables and none connection between them. Then, it uses a greedy-search to add, at each step, the arc that produces the graph structure with minimum entropy. It stops when the network structure B_S reaches a low *enough* entropy level. Kutaó can be written as algorithm 2.

Algorithm 2 Kutaó

```
Require: a database $D$ on variables $\mathbf{X} = \{X_1, \cdots, X_n\}$; a fixed value for entropy $\alpha$ and an
    order $Pred$ between variables
Ensure: a network structure $B_S$ (an oriented DAG)
1: Buil a structure on $\{X_1, \cdots, X_n\}$ and assume all variables to be marginally independent
2: $\beta := H(B_S)$ {Calculate the entropy of the structure}
3: repeat
4:    Select a link such that
        (a) it creates no cycle
        (b) is the one that creates a new structure $B_S$ with minimum entropy
        (c) links variables $X, Y$ such that $X$ comes first in the order
5:    Give the orientation $X \longrightarrow Y$
6: until $\beta \leq \alpha$
```

4.2 The Bayesian inference approach

We are interested in the most probable hypothesis (a Bayesian network in our case) from some space H given the observed data D plus any initial knowledge about the prior probabilities of the various hypotheses in H. Bayes theorem provides a direct method for calculation such probabilities. More precisely, Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself.

We shall write $P(h)$ to denote the initial probability that hypothesis h holds, before we may have observed the training data. $P(h)$ is often called the *prior probability* of h and may reflect any background knowledge we have about the chance that h is a correct hypothesis. If we have to such prior probability that training data D will be observed. Next, we will write $P(D|h)$ to denote the probability of observing data D given some world in which hypothesis h holds. In machine learning we are interested in the *posterior probability* of h, that is, the probability $P(h|D)$ that h holds given the training data D.

The Bayes theorem provides a way to calculate the posterior probability $P(h|D)$, from the prior probability $P(h)$, together with $P(D)$ and $P(D|h)$.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{11}$$

In machine learning, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis $h \in H$ given the observed data D. Usually, it is computationally unfeasible to find the most probable hypothesis, thereof some heuristical search strategies are used.

Note that in order to compare the posterior probabilities of two candidate hypotheses h, we actually do not need to calculate the term $P(D)$ of the Bayes theorem (see equation 11). This holds only when the posterior probabilities are calculated with respect the same data set D.

In some cases, we will assume that every hypothesis in H is equally probable a priori, that is, $P(h_i) = P(h_j)$ for all h_i and h_j in H. In this case we can also drop the term $P(h)$ from the Bayes theorem equation.

The reader may find a deep and an understandable introduction to the Bayesian approach of learning from data in the Heckerman's tutorials [15, 9].

4.2.1 The K2 algorithm

Cooper and Herskovits [8] designed a Bayesian method for constructing a probabilistic network from data. The algorithm searches for a probabilistic network structure with high posterior given a database of cases, and outputs the structure and its probability.

Now, we are going to develop (following [8]) the *scoring* function in order to measure the posterior $P(B_S|D)$ of the structure of a network $S = (B_S, B_P)$, where B_S denotes the network structure and B_P denotes the conditional probability assignments associated with the structure. Since we want to compare networks we will compute $P(B_{S_1}|D)/P(B_{S_2}|D)$

$$\frac{P(B_{S_1}|D)}{P(B_{S_2}|D)} = \frac{\frac{P(B_{S_1},D)}{P(D)}}{\frac{P(B_{S_2},D)}{P(D)}} = \frac{P(B_{S_1},D)}{P(B_{S_2},D)} \tag{12}$$

Thus, we will use $P(B_S, D)$ as a scoring function for measuring quality of network structures. To do so, we will introduce four assumptions:

Assumption 1: The database variables are discrete. This assumption allows us to use a probability mass function $P(D|B_S, B_P)$ rather than a density function

$$\begin{aligned} P(B_S, D) &= \int_{B_P} P(D|B_S, B_P) f(B_P|B_S) P(B_S)\, dB_P \\ &= P(B_S) \int_{B_P} P(D|B_S, B_P) f(B_P|B_S)\, dB_P \end{aligned} \tag{13}$$

where B_P is a vector where values denote the conditional probability assignements associated with the belief network structure B_S, and f is the conditional probability density function over B_P given B_S

Assumption 2: Cases of the database occur independently given a belief network S. This allows to calculate the probability of the whole dataset as the product of the probability of each case:

$$P(B_S, D) = P(B_S) \int_{B_P} \left[\prod_{h=1}^{m} P(c_h | B_S, B_P) \right] f(B_P | B_S) \, dB_P \tag{14}$$

where m is the number of cases in D and c_h is the h-th case in D.

Assumption 3: There are no cases that have variables with missing values. This allow us to calculate the probability of each case as the product of the probability of variables.

$$P(B_S, D) = P(B_S) \int_{B_P} \left[\prod_{h=1}^{m} \prod_{i=1}^{n} P(x_i^h | \mathbf{pa}_i^h, B_P) \right] f(B_P | B_S) \, dB_P \tag{15}$$

where x_i^h denotes the state of variable x_i in case c_h and pa_i^h denotes the configuration of the set of parents of the variable x_i in the network structure B_S in case c_h. By grouping terms, we can rewrite equation (15) as

$$P(B_S, D) = P(B_S) \int_{B_P} \left[\prod_{i=1}^{n} \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} P(x_i^k | \mathbf{pa}_i^j, B_P)^{N_{ijk}} \right] f(B_P | B_S) \, dB_P \tag{16}$$

where r_i stands for the number of states of variable x_i, x_i^k denotes the k-th state of the variable, q_i denotes the number of configurations of the parent set of x_i, $\mathbf{pa}_i^j$ denotes the j-th configuration of the variable x_i parents, and N_{ijk} denotes the number of cases in dataset D where $x_i = x_i^k$ and $\mathbf{pa}_i = \mathbf{pa}_i^j$.

Before going to the fourth assumption, let us introduce more notation: let θ_{ijk} be $P(x_i^k | \mathbf{pa}_i^j, B_P)$, and let call an assignment of numerical probabilities to $\theta_{ijk} \forall k = 1, \ldots, r_i$, a probability distribution, which we represent as the list $(\theta_{ij1}, \ldots, \theta_{ijr_i})$. Note that, since the states x_i^k are exclusive and exhaustive, it follows that $\sum_{k=1}^{r_i} \theta_{ijk} = 1$. In addition, for a given x_i and $\mathbf{pa}_i^j$, let $f(\theta_{ij1}, \ldots, \theta_{ijr_i})$ denote the probability density function over $(\theta_{ij1}, \ldots, \theta_{ijr_i})$

Assumption 4: Before observing D, we are indifferent regarding which numerical probabilities to assign to the belief network with structure B_S. From this assumption results the following two:

4a The distribution $f(\theta_{ij1}, \ldots, \theta_{ijr_i})$ is independent of the distribution $f(\theta_{i'j'1}, \ldots, \theta_{i'j'r_i})$, for $1 \leq i, i' \leq n$, $1 \leq j \leq q_i$, $1 \leq j' \leq q_{i'}$, and $ij \neq i'j'$;

4b Distribution $f(\theta_{ij1}, \ldots, \theta_{ijr_i})$ is uniform, for $1 \leq i, \leq n$, $1 \leq j \leq q_i$, that is , a non informative distribution.

From assumption 4a follows that

$$f(B_P | B_S) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} f(\theta_{ij1}, \ldots, \theta_{ijr_i}) \tag{17}$$

By substitution in equation (16) we obtain

$$\begin{aligned}P(B_S, D) = \quad & P(B_S) \int \overset{\theta_{ijk}}{\cdots} \int \left[\prod_{i=1}^{n} \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} \theta_{ijk}^{N_{ijk}} \right] \prod_{i=1}^{n} \prod_{j=1}^{q_i} f(\theta_{ij1}, \ldots, \theta_{ijr_i}) \\ & d\theta_{111}, \ldots, d\theta_{ijk}, \ldots, d\theta_{nq_nr_n}\end{aligned} \tag{18}$$

where the integral is taken over all $\theta_{ijk} \forall i \in [1, n]$, $\forall j = 1, \ldots, q_i$, and $\forall k = 1, \ldots, r_i$ such that $0 \leq \theta_{ijk} \leq 1$, and for every i, j the following condition holds $\sum_k \theta_{ijk} = 1$.

From the independence of terms in equation (18) follows

$$P(B_S, D) = P(B_S) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \int \overset{\theta_{ijk}}{\cdots} \int \left[\prod_{k=1}^{r_i} \theta_{ijk}^{N_{ijk}} \right] f(\theta_{ij1}, \ldots, \theta_{ijr_i})\, d\theta_{ij1}, \ldots, d\theta_{ijr_i} \tag{19}$$

By assumption 4b, it follows that $f(\theta_{ij1}, \ldots, \theta_{ijr_i}) = C_{ij}$ for some constant C_{ij}. Since $f(\theta_{ij1}, \ldots, \theta_{ijr_i})$ is a probability density function, it necessary follows that, for a given i and j,

$$\int \overset{\theta_{ijk}}{\cdots} \int C_{ij}\, d\theta_{ij1}, \ldots, d\theta_{ijr_i} = 1 \tag{20}$$

Now, we must solve this equation which is a special form of the Dirichlet's integral:

$$\int \overset{\theta_{ijk}}{\cdots} \int \prod_{i=1}^{r_i} \theta_{ijk}^{N_{ijk}}\, d\theta_{ij1}, \ldots, d\theta_{ijr_i} = \frac{\prod_{k=1}^{r_i} N_{ijk}!}{(N_{ij} + r_i - 1)!} \tag{21}$$

where $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$. If now, we take $N_{ijk} = 0$ (hyperparameters yielding a non informative distribution as required by assumption 4b) for all ijk we obtain:

$$\int \overset{\theta_{ijk}}{\cdots} \int 1\, d\theta_{ij1}, \ldots, d\theta_{ijr_i} = \frac{1}{(r_i - 1)!} \tag{22}$$

and thus,

$$\int \overset{\theta_{ijk}}{\cdots} \int (r_i - 1)!\, d\theta_{ij1}, \ldots, d\theta_{ijr_i} = 1 \tag{23}$$

and we can solve equation (20) obtaining $C_{ij} = (r_i - 1)!$ and therefore $f(\theta_{ij1}, \ldots, \theta_{ijr_i}) = (r_i - 1)!$ is a uniform probability density function. If we substitute this solution in equation (19)

$$\begin{aligned}P(B_S, D) \quad &= P(B_S) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \int \overset{\theta_{ijk}}{\cdots} \int \left[\prod_{k=1}^{r_i} \theta_{ijk}^{N_{ijk}} \right] (r_i - 1)!\, d\theta_{ij1}, \ldots, d\theta_{ijr_i} \\ &= P(B_S) \prod_{i=1}^{n} \prod_{j=1}^{q_i} (r_i - 1)! \int \overset{\theta_{ijk}}{\cdots} \int \prod_{k=1}^{r_i} \theta_{ijk}^{N_{ijk}}\, d\theta_{ij1}, \ldots, d\theta_{ijr_i}\end{aligned} \tag{24}$$

Note that the multiple integral in equation (24) is again the Dirichlet's integral (equation 21) and we can substitute it in equation (24) obtaining finally the scoring function $P(B_S, D)$:

$$P(B_S, D) = P(B_S) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{25}$$

Once we have the scoring function we only have to find the network structure that maximizes that function. We assume that, before observing the dataset D, we belief that all structures are equally likely. Therefore, the prior $P(B_S)$ turns in an uniform probability distribution (i.e. a constant c) which we can skip as it does not play any role in maximizing the scoring function:

$$\max_{B_S}[P(B_S, D)] = \prod_{i=1}^{n} \max_{\mathbf{Pa}_i} \left[\prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \right] \tag{26}$$

where the maximization on equation (26) takes place over every instantiation of the parents $\mathbf{Pa_i}$ of the variable X_i. We also must be aware of no allowing cycles in the network structure B_S when maximizing the scoring function. In order to avoid cycles, we introduce an order among the n variables, such that, if X_i precedes X_j in the ordering, X_j cannot be parent of X_i.

Cooper and Herskovits propose an heuristic greedy search that begins with the assumption that a node has no parents, and then adds incrementally that parent whose addition most increases the probability of the resulting structure. When the addition of no single parent can increase the probability, it stops adding parents to the node. In particular, the algorithm uses the following scoring function which follows from equation (26)

$$g(X_i, \mathbf{Pa}_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{27}$$

where the N_{ijk} are computed relative to $\mathbf{Pa}_i$ and to the database D.

The **K2** can be written as algorithm 3.

Algorithm 3 K2

```
Require: a database $D$ on the variables $\{X_1, \ldots, X_n\}$, an order $Pred$ among the variables,
    and a maximum number of parents per node $u$
Ensure: a DAG with maximum a posteriori probability given the database $D$
 1: Build a structure $B_S$ on $\{X_1, \cdots, X_n\}$ and assume all variables to be marginally indepen-
    dent
 2: for $i := 1$ to $n$ do
 3:   $\mathbf{Pa}_i := \emptyset$
 4:   $P_{old} := g(X_i, \mathbf{Pa}_i)$
 5:   OK := true
 6:   while OK and $(\mathbf{Pa}_i < u)$ do
 7:     let $z$ be the node in $Pred(X_i) - \mathbf{Pa}_i$ that maximizes $g(X_i, \mathbf{Pa}_i \cup \{z\})$
 8:     $P_{new} := g(X_i, \mathbf{Pa}_i \cup \{z\})$
 9:     if $P_{new} > P_{old}$ then
10:       $P_{old} := P_{new}$
11:       $\mathbf{Pa}_i := \mathbf{Pa}_i \cup \{z\}$
12:     else
13:       OK := false
14:     end if
15:   end while
16: end for
```

Cooper and Herskovits also proposed a closed formula in order to calculate the expectations of the conditional probabilities when given a database D and a Bayesian network structure B_S.

Let denote θ_{ijk} the conditional probability $P(x_i^k|Pa_i^j)$, that is, the probability that X_i has value x_i^k, for some k from 1 to r_i, given that the parents of X_i has the configuration Pa_i^j. Call θ_{ijk} a network conditional probability and let ξ denote the four assumptions.

Consider the value of $E[\theta_{ijk}|D, B_S, \xi]$ be the expected value of θ_{ijk} given the database D, the Bayesian network structure B_S and the assumptions ξ:

$$E[\theta_{ijk}|D, B_S, \xi] = \frac{N_{ijk}+1}{N_{ij}+r_i} \tag{28}$$

This equation is fully justified in [8].

4.3 The Minimum Description Length approach

In this section we introduce the Rissanen's Minimal Description Length (MDL) principle. The MDL principle is based on the idea that the best model of a database is the model that minimizes the sum of the length of the encoding of

1. The model
2. The data given the model

In order to apply the MDL principle we first need to encode the Bayesian network, as the model, and the raw data given the network. Afterwards, we will measure in bits the length of both encodings.

1. Encoding the Network

A Bayesian network $S = (B_S, B_P)$ is formed by the structure B_S (i.e. a DAG) and a list of the conditional probabilities B_P associated to each node. So, in order to encode the network we need to encode its structure and the set of conditional probabilities.

Suppose there are n nodes in the dataset. For a node X_i with $|\mathbf{Pa_i}|$ different parents, we need $|\mathbf{Pa_i}| \log_2(n)$ bits to list its parents. Thus, we need the following number of bits in order to encode the structure of a network

$$\sum_{i=1}^{n} |\mathbf{Pa_i}| \log_2(n)$$

The encoding length of the conditional probabilities for each node X_i is the product of the number of bits required to store the numerical value of each conditional probability and the total number of conditional probabilities that are required. Thus, we need the following number of bits in order to encode the conditional probabilities of a network

$$\sum_{i=1}^{n} d\,(r_i - 1)\, q_i$$

where r_i is the number of states of the node X_i, q_i is the number of configurations of its parents and d the number of bits required to store a numerical value. Note, that since the states of the nodes are independent and exclusive, the sum over all values of conditional probabilities equals 1, and therefore we only need $(r_i - 1)\, q_i$ numbers (instead of $r_i\, q_i$) to fully specify the conditional probabilities. Note also, that for a given dataset D, n and d are constant.

Finally, the encoding length of a Bayesian network is the sum of the two values previously stated

$$\sum_{i=1}^{n} |\mathbf{Pa_i}| \log_2(n) + d\,(r_i - 1)\, q_i \tag{29}$$

2. Encoding the data given the model

Here, we want to encode the dataset D of m cases, given the model $S = (B_S, B_P)$. Since we are interested in comparing the length of encoding the data given a Bayesian network, we actually do not care in using the most efficient code. We will use the *character codes* which is intuitive and not very time consuming. The *character codes* assigns to each configuration a unique binary string, and the dataset D is encoded by concatenating the m binary strings of the cases. It is well known that we can minimize the length of the final binary string by giving the shortest code to the cases with the highest frequency. We minimize the coding length by applying the **Huffman algorithm**:

> If we want to encode an alphabet A, construct the code backward starting from the tail of the codewords,
>
> 1. Take the two least probable symbols in the alphabet. These two symbols will be given the longest codewords, which will have equal length, and differ only in the last digit.
> 2. Combine these two symbols into a new single symbol, calculate the probability of the new symbol, and repeat.
>
> Since each step reduces the size of the alphabet by one, this algorithm will have assigned strings to all the symbols after $|A| - 1$ steps.

Huffman's algorithm requires as input the frequency of occurrence of each configuration appearing in the database. Suppose that each configuration c_i in the database has probability p_i, then Huffman's algorithm assigns to configuration c_i a codeword of length approximately $-\log_2(p_i)$. If we have m, being m large, in the database, then the length of the string encoding the database is approximately

$$-m \sum_i p_i \log_2(p_i) \tag{30}$$

where we are summing over all possible configurations.

Evidently, we do not have these p_i probabilities since the Bayesian network is a guess (or model) of such probabilities. The Bayesian network, as a model, assigns a probability q_i to each configuration. Of course, in general q_i is not equal p_i although it is the aim for q_i to be close to p_i. The closer q_i to p_i the more accurate is the Bayesian network.

We will use the probabilities q_i given by the Bayesian network to compute the Huffman code of dataset D. Hence, each configuration c_i is assigned a codeword of length approximately $-\log_2(q_i)$ and the length of the string encoding the database is approximately

$$-m \sum_i p_i \log_2(q_i) \tag{31}$$

Now we can use the **Gibbs' theorem** in order to compare equations (30) and (31):

Theorem 1 *Let p_i and q_i, where $i = 1, \ldots, t$, be non-negative real numbers that sum to 1. Then,*

$$-m \sum_{i}^{t} p_i \log_2(p_i) \leq -m \sum_{i}^{t} p_i \log_2(q_i)$$

with equally holding iff $\forall i, p_i = q_i$. In the summation we take $0 \log_2(0)$ to be 0.

From this theorem, it comes out that the encoding using the estimated probabilities q_i is longer than the encoding using the true probabilities p_i. It also says that the true probabilities achieve the minimal encoding length.

The MDL principle says that we must choose the network that minimizes the encoding length of the dataset, and we have seen that it depends on the accuracy of the network. We can use equation (31) in order to evaluate the encoding length of the dataset given the network. However, this measure has two problems. First, we do not know the values of p_i, and second, equation (31) requires a summation over all the configurations, and the number of configurations is exponential in the number of variables.

The first problem is easily overcome by the *law of large numbers*: the configuration c_i with probability p_i is expected to appear $m \cdot p_i$ times in a database of m cases. Hence, we can use c_i's frequency in to the database as an estimator of p_i.

In order to overcome the second problem, we can use Gibbs' theorem to relate the encoding length of the data to the Kullback-Leibler divergence $D_{KL}(P, Q)$ (equation 8). We already know that $0 \leq D_{KL}$ and that is zero if and only if both probability distributions (P and Q) are identical. From equation (31), the Kullback-Leibler divergence $D_{KL}(P, Q)$ (equation 8) and Gibbs' theorem we have the following theorem:

Theorem 2 *The encoding length of the data is a monotonically increasing function of the Kullback-Leibler divergence between the distribution defined by the model and the true distribution.*

This theorem shows that instead of using the data encoding length (equation 31) we can use the Kullback-Leibler divergence to evaluate candidate networks. Although the Kullback-Leibler divergence also involves a summation over an exponential number of configurations, a computational feasible approach for evaluating this measure can be developed by extending the work of Chow and Liu [7] as we will see in the next section.

For further readings on the Minimum Description Length principle see [27, 28].

4.3.1 The Lam and Bacchus algorithm

Lam and Bacchus [19] extended the Chow and Liu [7] work to a more general result. Chow and Liu proposed a measure (see equation 9) to measure the difference between the dataset distribution P and the tree dependence distribution P_τ which models the dataset. Lam and Bacchus extended the measure to the general case of a network structure, i.e. directed acyclic graph (DAG), with dependence distribution P_S.

Chow and Liu demonstrated that minimizing $D_{KL}(P, P_\tau)$ was equivalent to maximize the mutual information (equation 7) $\sum_{i=1}^{n} I(X_i; X_{j(i)})$ where $X_{j(i)}$ represents the *unique parent* of the node X_i. In the same way, Lam and Bacchus demonstrated that minimizing $D_{KL}(P, P_S)$ is equivalent to maximize the mutual information $\sum_{i=1}^{n} I(X_i; \mathbf{Pa}_i)$ where $\mathbf{Pa}_i$ is the *set of parents* of the node X_i. In conclusion, given the probabilities computed from the dataset, we can calculate the quality of various candidate networks using local computation at each node. Thus, if we can find a network with maximum total mutual information, the probability distribution of this structure will be the one closest to the underlying distribution of the dataset. And, by theorem 2, it will yield the shortest encoding of the data.

Nevertheless, if we used this measure as a quality measure, we would obtain the multiply connected network corresponding to the underlying probability of the data set. This structure would be a complete graph with a large encoding, and worse still, it would not convey any information since it could represent any distribution.

The MDL principle, anyway, allows to perform a trade-off between complexity of the structure and its accuracy. The totally connected network would require to store an exponential number of probability parameters. Hence, there will probably be a less complex network with a shorter encoding that is still able to produce a reasonably short encoding of the data. When evaluating the total description length, this less complex network is preferred.

Now, we proceed to describe the heuristical search proposed by Lam and Bacchus [19]. Given a dataset D on $\mathbf{X} = \{X_1, \ldots, X_n\}$ variables, we want to construct a Bayesian network with n nodes and a reasonable high score for MDL. We know that a directed acyclic graph with n nodes can have from 0 to $n(n-1)/2$ arcs between nodes. The process maintains $n(n-1)/2+1$ separate search sets S_i for candidate networks with i arcs between nodes. Each element of the set S_i has two components, namely, a candidate network B_S with i arcs and a pair of nodes (X_i, X_j) between which a new arc could be added to the candidate network without creating a cycle. Also, the elements of each set S_i are separated into distinct lists, the $OPEN_{S_i}$ and the $CLOSED_{S_i}$ list. The algorithm performs a best-first search within each set S_i using the $OPEN_{S_i}$ and $CLOSED_{S_i}$ lists. See algorithm 4.

The algorithm calls the procedrue PD-procedure(S,(X_i, X_j)) that adds an arc between the nodes X_i and X_j creating a new network S'. It chooses the direction of the new arc that most increases the network's accuracy (i.e. mutual information). In the process it might also reverse the direction of the other arcs in B_{Sold}. See procedure 5.

5 Learning Bayesian Networks: the incremental approach

In this section we consider the main contributions about incremental or on-line learning algorithms for Bayesian networks. The aim of all these algorithms is to modify or evolve an already known structure when new data is available. However, in the learning Bayesian networks field, there is neither a wide accepted definition of what is considered to be an incremental algorithm, nor of which is the aim of such algorithms. Friedman and Goldszmidt are the only authors, as far as we know, to give a precise definition of on-line or incremental algorithms in the Bayesian network learning field [12]:

> **Definition:** A Bayesian network learning procedure is incremental if at each iteration l, it receives a new data instance u_l and then produces the next hypothesis S_{l+1}. This estimate is then used by to perform the required task (i.e. prediction, diagnosis, classification, etc.) on the next instance n_{l+1}, which in turn is used to update the network and so on. The procedure might generate a new model after some number of k instances are collected.

If we compare this definition to that given by Langley (see Section 2.3), we can see that it is the same from the viewpoint of the behavior of the process. Both definitions require the

Algorithm 4 Lam and Bachus

```
Require: a database $D$ on the variables $\{X_1, \ldots, X_n\}$
Ensure: $B_{Sres}$ be a DAG with low Minimum Descritption Length given the database $D$
 1: Calculate the mutual information $I(X_i; X_j)$ between every pair of distinct nodes
 2: PAIRS = list of pairs of nodes. Ordered from the highest to the lowest mutual information.
 3: Initialize all $S_i$ search sets with $i = 1 \ldots n(n-1)$ as empty sets.
 4: Initialize $S_0$ containing the single element with the candidate network with no arcs and
    the first pair of nodes from the PAIR list. Let this element be in the $OPEN_{S_0}$ list.
 5: repeat
 6:   for each set $S_i$ do
 7:     Remove the element with greatest heuristic value, $\sum_{i=1}^{n} I(X_i; \mathbf{Pa}_i)$, from the
        $OPEN_{S_i}$ list and copy it onto the $CLOSED_{S_0}$ list. Let the element's network
        be $B_{Sold}$ and the element's pair of nodes be $(X_i, X_j)$.
 8:     $B_{Snew}$ = PD-procedure($B_{Sold}$,$(X_i, X_j)$)
 9:     If $B_{Snew}$ is fully connected, place a copy of it into a set, FINAL, of final candidates.
10:     Build a new search element consisting of $B_{Snew}$ and the first pair of nodes from
        PAIRS that appears after the old pair $(X_i, X_j)$ and between which an arc could be
        added without generating a cycle in $B_{Snew}$. Insert this element into the $OPEN_{S_{i+1}}$
11:     Build a new search element consisting of $B_{Sold}$ and the first pair of nodes from PAIRS
        that appears after the old pair $(X_i, X_j)$ and between which an arc could be added
        without generating a cycle in $B_{Snew}$. Insert this element into the $OPEN_{S_i}$ list.
12:   end for
13: until a limited amount of computational resources is been used in each $S_i$
14: Let $B_{Sres}$ be the structure from the FINAL set with the highest MDL score.
```

Procedure 5 PD-procedure

```
Require: a network $S$ and a pair of nodes $(X_i, X_j)$
Ensure: $S'$ be a network with an arc between $X_i$ and $X_j$ and posibly some other arcs reversed
 1: for all arc $\in \{X_i \longrightarrow X_j, X_i \longleftarrow X_j\}$ do
 2:   Create a new network by adding the arc
 3:   Determine the optimal directionality of the arcs attached directly to $X_j$ by examining
      which directions maximize the mutual information.
 4:   if the direction of an existing arc is reversed then
 5:     perform the above directionality determination step on the other node afected
 6:   end if
 7: end for
 8: $S'$ = network of greatest mutual information from the two networks found
```

learning algorithm to process training instances as they are available and to have a domain model ready for performance tasks at each iteration. However, Friedman and Goldszmidt's definition, does not state any restriction on the way the learning algorithm process data. Thus, an algorithm that reprocess the whole data set at each iteration or that maintains lots of alternative Bayesian networks in memory fits to this definition.

Anyway, Friedman and Goldszmidt's definition introduce an interesting point. Namely, first it allows some granularity for the incremental process, that is, the incremental process does not have to yield a new domain model for each single data item. Instead, it is allowed to wait until some number k of data instances are available, so we can say that the process is incremental with some granularity.

Up to 1998, there are only three proposals of algorithms revising the network structure of a Bayesian network. Namely, Buntine's (1991) [4], Lam and Bacchus' (1994) [20] and Friedman and Goldszmidt's (1997) [12]. The algorithm proposed by Wray Buntine yields a set of alternative and reasonable networks given the dataset. The algorithm is able to revise the set of Bayesian networks in the light of new data. Lam and Bacchus proposed an algorithm able to revise parts (i.e. a subgraph) of the already learnt Bayesian network when new data about a subset of variables is available. Finally, Friedman and Goldszmidt proposed an algorithm that explores a *frontier* of possible alternative networks. When new data is presented to the algorithm the frontier is changed and the best network is selected as the result.

We present these three algorithms in a chronological order. It is difficult to compare the results obtained by the three proposals for two reasons. Firstly, only Friedman and Goldszmidt present some results in their article, and secondly, we think it cannot be said that the aim of all of them is exactly the same.

5.1 Buntine's proposal

Wray Buntine [4] proposed an incremental algorithm for learning Bayesian Networks. The algorithm, given a dataset and a total ordering of the variables, comes up with different alternative Bayesian networks that are reasonable in terms of the scoring functions. We call *reasonable* those alternative Bayesian networks whose scorings are within a factor E of the best found, where E is a parameter of the algorithm. Note that, in some way, the parameter E also specifies the number of alternative networks the algorithm yields.

The capability of the algorithm to return a list of alternative Bayesian networks contrasts with the algorithms we have seen so far in this report as they yield one single network.

Buntine proposes first a batch algorithm that uses the Bayesian approach for the scoring functions. Afterwards, he proposes some guidelines for converting it into an incremental or on-line algorithm. The batch algorithm can be seen like a generalization of K2 since when the factor E is set to 1 the algorithm is actually that proposed by Cooper and Herskovits [8]. Furthermore, the incremental version of the algorithm can also be seen as another generalization.

Here, we follow Buntine [4] stating first the batch version of the algorithm and afterwards the incremental one.

5.1.1 The batch version of the algorithm

The algorithm needs a total ordering of the variables as prior knowledge from the experts. The variables coming first in the ordering are supposed to influence the others. We need also a compact data structure in order to represent the alternative Bayesian networks.

For each variable X_i we will keep a set of reasonable alternative parent sets Π_i according to some criteria of reasonableness. For the variable X_i alternative parent sets $\Pi_i = \{\mathbf{Pa}_{i1}, \ldots, \mathbf{Pa}_{im}\}$ will be a collection of subsets of $\{Y : Y \prec X_i\}$. We also have to store the network parameters θ_{ijk} for each set of possible parent sets. The space of alternative networks is then given by the Cartesian product across the sets of the parent sets for each variable $\otimes_{i=1}^{n} \Pi_i$. Buntine calls this structure, both the set of parent sets and the network parameters, a *combined Bayesian network.*

In order to access all alternative parent sets $\mathbf{Pa}_i \in \Pi_i$ efficiently they are stored in a lattice structure where the subset and superset parent sets are linked together in a web, denoted the parent lattice for X_i. Since the full set of lattices is of size potentially exponential to the number of variables n, only those parent sets with significant posterior probabilities are stored and linked.

The parent lattice Π_i for the node X_i is stated as follows. The root node is the empty set and the leaves are the sets $\mathbf{Pa}_i$ which have no supersets contained in Π_i. For example, the lattice $\Pi_i = \{\{a\}, \{a,b\}, \{a,c\}, \{a,d\}\}$ has the root $\{a\}$ and the leaves $\{a,b\}, \{a,c\}$ and $\{a,d\}$. The number of leaves can be reduced by adding the parent sets $\{a,b,c\}, \{a,c,d\}$ and $\{a,c,d,e\}$, resulting in a lattice with the leave $\{a,c,d,e\}$.

The algorithm has tree parameters $G < F < E < 1$ which, in some way, specify the sort of search will be performed. When the parameters are close to 1, the search becomes greedy since the number of alternative networks is reduced and, on the contrary, when the parameters are close to 0, the search is a beam search. The closer to zero the parameters are, the more beams the algorithm considers during the search process.

According to the parameters the algorithm classifies the parent sets as alive, dead or asleep. Alive parent sets represent the set of reasonable alternatives having posteriors within a factor of E of the best found. Dead parent sets exist in the lattice as dead-end markers in the search space. They have been explored and forever determined to be unreasonable alternatives and are not to be further explored. Asleep parent sets are similar but are only considered unreasonable for now and may be made alive later on. Furthermore, nodes can be either open or closed, depending on whether they require further expansion during search.

Buntine uses the Bayesian approach in order to calculate the scoring functions, although he does not define the hyper-parameters of the Dirichlet distribution. In order to calculate the posterior probabilities we can use the uniform Dirichlet distribution proposed by Cooper and Herskovits and use their function (equation 27). Buntine's proposa is stated as in algorithm 6, where the process-the-children(ch_i) (procedure 7) updates the lists of nodes according to the posterior of the parent sets $\mathbf{Pa}_i \in ch_i$

We want to remark, here, that when the parameters E, F and G are set to 1, this algorithm reduces to the K2 algorithm. Note that in such situation only one network structure is kept into the open-list and all the other lists are empty. Thus, we can think of this algorithm like a generalization of the K2 algorithm. However, recall that Buntine's algorithm yields a combined Bayesian network rather than a single network.

Algorithm 6 Buntine's batch

```
Require: a database $D$ on the variables $\{X_1, \ldots, X_n\}$, an order $\prec$ among variables, and the parameters $G < F < E < 1$, and a database $D$
Ensure: a combined Bayesian network corresponding to reasonable alternatives according to the parameters.
 1: for $i = 1$ to $n$ do
 2:   $\textit{Best-posterior} = P(\mathbf{Pa}_i = \emptyset | D, \prec)$
 3:   $\textit{Open-list} = \{\emptyset\}$ {Parent sets within a factor $E$ of Best-posterior, those to be further expanded.}
 4:   $\textit{Alive-list} = \{\emptyset\}$ {Parent sets within a factor $F$ of Best-posterior}
 5:   repeat
 6:     Take from the Open-list the parent set $\mathbf{Pa}_i$ with the highest posterior $P(\mathbf{Pa}_i|D, \prec)$
 7:     if $P(\mathbf{Pa}_i|D, \prec) < G \cdot \textit{Best-posterior}$ then
 8:       Mark this parent set as dead
 9:     else if $P(\mathbf{Pa}_i|D, \prec) < F \cdot \textit{Best-posterior}$ then
10:       ignore this parent set
11:     else
12:       generate all its children $ch_i$ and calculate their posterior
13:       process-the-children($ch_i$)
14:     end if
15:   until $\textit{Open-list} = \{\emptyset\}$
16: end for
```

Procedure 7 Process-the-children

```
Require: the children sets of parents $ch_i$
Ensure: lists of nodes updated
  $Pa_i$ = the parent set rom $ch_i$ with the highest posterior $P(\mathbf{Pa}_i|D, \prec)$
  if $P(\mathbf{Pa}_i|D, \prec) > \textit{Best-posterior}$ then
    $\textit{Best-posterior} = P(\mathbf{Pa}_i|D, \prec)$
    modify the Alive-list to reflect the new maximum
  end if
  for all children $\mathbf{Pa}_i \in ch_i$ do
    if $P(\mathbf{Pa}_i|D, \prec) < G \cdot \textit{Best-posterior}$ then
      mark $\mathbf{Pa}_i$ as dead
    else if $P(\mathbf{Pa}_i|D, \prec) > F \cdot \textit{Best-posterior}$ then
      $\textit{Alive-list} = \textit{Alive-list} \cup \{\mathbf{Pa}_i\}$
    else if $P(\mathbf{Pa}_i|D, \prec) > E \cdot \textit{Best-posterior}$ then
      $\textit{Open-list} = \textit{Open-list} \cup \{\mathbf{Pa}_i\}$
    end if
  end for
```

5.1.2 The incremental version of the algorithm

Buntine proposes the incremental algorithm as an extension of the one seen above. He considers the situation where new data are available and have to be processed by the algorithm in order to update the combined network.

Buntine describes two different situations depending on the time available in order to update the combined network. Namely, one when the algorithm can spend very few time for updating, and another when more time is available. In the first case, a rapid update of the combined network is required and there is no time enough to update the parent structure. Thus, the algorithm only updates the posterior probabilities of the parent lattices. In the second case, given additional time, both structure and posteriors update are performed.

In order to update the posteriors of the combined network, we need to store posterior probabilities and the counters N_{ijk} for each alternative set of parent sets in order to be able to update them when new information is available. We also need an incremental expression of $g(X_i, \mathbf{Pa}_i)$ (equation 27). Suppose that the dataset D is extended to database D' with the new example having $X_i = x_i^k$ and $\mathbf{Pa}_i = j$, then we should increment N_{ijk} and calculate the new posteriors $g'(X_i, \mathbf{Pa}_i)$ as

$$g'(X_i, \mathbf{Pa}_i) = g(X_i, \mathbf{Pa}_i) \frac{N_{ijk}}{N_{ij} + r_i - 1} \tag{32}$$

This follows from the recursive properties of the Gamma (factorial) function. Since the number of reasonable parents is $L = \sum_{i=1}^{n} |\mathbf{Pa}_i|$, the full update process will therefore take $O(L)$ operations. If we increase D by adding I new examples in a batch then we can repeat this process I times.

When a batch of new examples and additional time is available, a process reproducing the results of the algorithm can be run incrementally. In each step of the **for** iterative structure of the main algorithm, that is, for each variable X_i of the already discovered combined Bayesian network the following must be done:

1. Update the posterior probabilities of all alive and open parent sets of the lattice.

2. Calculate the new *Best-Posterior*

3. Expand nodes from the *Open-list* and continue with the search

It may happen that some parent sets oscillate on and off *Alive-list* and *Open-list* because the posteriors ordering of the parent set oscillate as the training examples are taking in account. This effect can easily be prevented by making a differential on E and F between placing a node on and taking a node off.

Note that this incremental algorithm is again a generalization of the former.

5.2 Lam and Bacchus' proposal

Lam and Bacchus [20] proposed an extension of their batch algorithm so that it could perform revision of the Bayesian network structure incrementally as new data is available.

The new refined network structure they want to discover should be similar to the existent one since the task they want to carry out is refinement. The refinement is done with the implicit assumption that the existent network is already a fairly accurate model of the database.

They demonstrate that if we improve the description length DL of a subgraph changing its topology, we improve the description length of the complete graph if we do not introduce cycles.

Theorem 3 *Let $B_{Sp} = (N_p, A_p)$ and $B'_{Sp} = (N_p, A'_p)$ be respectively two subgraphs of $B_S = (N, A)$ and $B'_S = (N, A')$, where N is the set of nodes and A is the set of arcs, and where $N_p \subseteq N$, $A_p \subseteq A$ and $A'_p \subseteq A'$. The following holds*

$$DL(B'_{Sp}) < DL(B_{Sp}) \Rightarrow DL(B'_S) < DL(B_S)$$

Using this theorem they developed an algorithm that improves the Bayesian network by improving parts of it. The algorithm first learns a new partial structure from the new data and the existent network using an extension of the minimum description length (MDL) measure, and then modifies locally the global old structure using the newly discovered partial structure. The new data is presented as a table of examples where only a subset of the variables represented by the Bayesian network are available.

5.2.1 Learning the partial structure

Recover from Section 4.3 that the MDL principle states that the best model of a database is the model that minimizes the sum of the length of encoding the model and the length of encoding the database given the model.

For the refinement problem, the source data consists of two components, the new data and the existent network structure. Thus, we must find a partial network B_{Sp} that minimizes the sum of the of the length of the encoding of

1. The partial network B_{Sp}
2. The new data given the network B_{Sp}
3. The existent network given the network B_{Sp}

Note that the sum of the last two items corresponds to the description length of the source data given the model. We are assuming that these two items are independent of each other given B_{Sp}, and thus they can be evaluated separately.

In order to calculate the encoding length of the first two items we will use the equations used in the section 4.3.1. For calculating the length of the encoding of the third item we need to compute the description of the complete existent network B_S given the partial one B_{Sp}. To recover B_S having B_{Sp} we only need to describe the differences between B_S and B_{Sp},

- a listing of reversed arcs, that is, those arcs in B_{Sp} that are also in B_S but with opposite direction
- the additional arcs of B_S, that is, those arcs in B_S that are not in B_{Sp}
- the missing arcs of B_S, that is, those arcs in B_{Sp} but are missing in B_S

A simple way to encode an arc is to describe the source node and the destination node. If we have n nodes we need $\log n$ bits to identify one. Therefore we need $2 \log n$ bits in order to describe an arc. Let r, a and m be respectively the number of reversed, additional and missing arcs in B_S with respect B_{Sp}. The description length B_S given B_{Sp} is then

$$(r+a+m)2\log n \tag{33}$$

This description length can be localized for each node. See that each arc can be uniquely assigned to its destination node. For a node X_i in B_S let r_i, a_i and m_i be the number of reversed, additional and missing arcs of it given B_{Sp}. If B_S structure is defined over the set $\mathbf{X}$ of nodes and the partial structure B_{Sp} is defined over $\mathbf{X}_p \in \mathbf{X}$ set of nodes, equation 33 can easily be reformulated as

$$\sum_{X_i \in \mathbf{X}} (r_i + a_i + m_i) 2 \log n \tag{34}$$

If $\mathbf{X}_q = \mathbf{X} \backslash \mathbf{X}_p$ then the sum of previous equation can be expressed as

$$\sum_{X_i \in \mathbf{X}_p} (r_i + a_i + m_i) 2 \log n + \sum_{X_i \in \mathbf{X}_q} (r_i + a_i + m_i) 2 \log n \tag{35}$$

The second sum in the above equation specifies the description lengths of the nodes $\mathbf{X}_q$ which are not present in B_{Sp}. Thus, the corresponding r_i's and m_i's are zero and the a_i's are not affected by the B_{Sp} structure. As we are using the measure in order to compare different partial structures B_{Sp}, the second part of the sum of equation 35 is constant over them. Therefore, we only need to compute the first part.

Finally, in order to learn the local structure we can use the batch algorithm proposed by Lam and Bacchus (see Section 4.3.1), and as scoring function for each node of the partial structure

$$DL_i = |Pa_i| \log n + \sum_{X_j \in Pa_i} I(X_i; X_j) + (r_i + m_i + a_i) 2 \log n \tag{36}$$

where the first term corresponds to the encoding length of the partial structure, the second corresponds to the encoding length of the new data given the partial structure and the last corresponds to the encoding length of the old structure given the new. This equation is banked on the results obtained by Lam and Bacchus [19] (see Section 4.3.1)

5.2.2 Modifying the global old structure

Suppose the existent network structure is B_S, and the learned partial structure is B_{Sp}. The objective of the refinement process is to obtain a refined structure of lower total description length with the aid of the existent structure B_S and the partial structure B_{Sp}.

Say we have a node X_i, its parent set in B_{Sp} is $Pa_i(B_{Sp})$, and its description length in B_{Sp} is DL_i (equation 35). In the existent network B_S, however, X_i will in general have a different set of parents $Pa_i(B_S)$ and a different description length. If $Pa_i(B_S) \not\subset \mathbf{X}_p$, then these two descriptions lengths are incomparable. In this case X_i has a parent in B_S that does not appear in the new data; hence the new data cannot tell us anything about the effect of that parent on X_i's description length. We identify all of the nodes X_i whose parents in B_S are also in B_{Sp} and call these the set of *marked* nodes.

Suppose for a certain marked node X_i, we decide to substitute the parents of X_i in B_S with the parents of X_i in B_{Sp}, a new structure B_{S1} is obtained. Usually the total description length of B_{S1} can be calculated simply by adding the total description length of the old structure B_S to the difference between the local description lengths of X_i in B_S and B_{Sp}.

The new total description length of B_{S1} can be evaluated in this way if the substitution of the parents of X_i in B_S does not affect the local description lengths of any other node in B_S. In fact, the only situation where this condition fails is when the parents of X_i in B_S contain a reversed arc (as compared to B_S). Under this circumstance, we need to consider the node X_r associated with this reversed arc. If X_r is also a marked node, we need to re-evaluate its local description length since it will be affected by the substitution of X_i's parents. Recursively, we must detect any other marked nodes that are, in turn, affected by the change in X_r's description length. It can be easily observed that these affected nodes must all be connected. As a result, we can identify a *marked subgraph unit* that contains only marked nodes and which can be considered together as an unit when the replacement is performed.

Actually, we can obtain the same subgraph unit if we had started off at any node in the subgraph due to the symmetrical nature of the influence between the nodes in the subgraph. For instance, returning to the previous example, if we considered X_r first, we would have detected that the local description length of X_i would be affected by the substitution of X_r's parents. The process would have continued and we would have obtained the same subgraph.

The following algorithm identifies a marked subgraph unit with respect to B_S. Initially, Q is a set containing the initial node X_i and it grows as the algorithm progresses. Q will contain the required marked subgraph when the algorithm (algorithm 8) terminates. Initially, M is a set containing some nodes that could be transferred to Q. It shrinks as the algorithm progresses and contains the remaining marked nodes that are not included in Q.

Procedure 8 Subgraph-unit

```
Require: $Q$, $X_I$, $M$
Ensure: $Q$ = set of marked nodes
1: $R$ = the set of reversed arcs from $X_i$'s parent set in $B_{Sp}$
2: for all $X_r \in R$ do
3:     $M = M - \{X_r\}$
4:     if $X_r$ is marked and $X_r \notin Q$ then
5:         $Q = Q \cup \{X - r\}$
6:         $subgraph - unit(Q, X_i, M)$
7:     end if
8: end for
```

Now, we can identify all marked subgraph units in B_{Sp} (see algorithm 9. Parent substitution is to be done for all the nodes in the subgraph if the subgraph is chosen for refinement. A useful property of the subgraph is that the change in description length of each subgraph is independent of all other subgraphs. The next algorithm identifies all marked subgraph units in B_{Sp}. Initially M contains all of the marked nodes and $S = \emptyset$. All subgraph units will be contained in S when the algorithm terminates. Q is a local variable containing the nodes for the current subgraph unit.

The refinement problem now is reduced to choosing appropriate subgraphs for which we should perform parent substitution in order to achieve a refined structure of lowest total description length. Although each subgraph substitution yields an independent reduction in description length, these substitutions cannot be performed independently as cycles may arise.

We use best-first search to find the set of subgraph units that yields the best reduction in description length without generating any cycles. To assist the search task, we construct

Procedure 9 Partition-into-subgraph

```
Require: $M$, $S$
Ensure: $S$ be a set of subgraph units
  while $M \neq \emptyset$ do
    $X_i$ is a node from $M$
    $M = M - \{X_i\}$
    $Q = \{X_i\}$
    $subgraph - unit(Q, X_i, M)$
    $S = S \cup \{Q\}$
  end while
```

a list $\mathbf{S} = \{S_1, S_2, \ldots, S_t\}$ by ranking all subgraphs in ascending order of the benefit gained if parent substitution was performed using that subgraph. The OPEN list contains search elements which consists of two components (B_S^r, S) where B_S^r is a refined network and S the next subgraph unit to be substituted into B_S^r. The elements in the OPEN list are ordered by the sum of the description length of B_S^r and the benefit contributed by the subgraph unit S. The initial OPEN list consists of the search elements (B_{Si}^r, S_{i+1}) where B_{Si}^r is obtained by substituting S_i into the existent structure B_S for $i = 1$ to $t - 1$.

Procedure 10 Find-subgraphs

```
Require: OPEN list
Ensure: OPEN list contains refined networks
  repeat
    Extract the first element from the OPEN list.
    Let it be $(B_S, S_i)$. Put $B_S$ into the CLOSED list.
    Construct a new refined structure $B_S^r$ by incorporating $S_i$ into $B_S$.
    Insert the element $(B_S, S_{i+1})$ into the OPEN list.
    if $B_S^r$ is acyclic then
      insert the element $(B_S^r, S_{i+1})$ into the OPEN list
    end if
  until resource limits are exceeded
```

5.3 Friedman and Goldszmidt's proposal

Friedman and Goldszmidt [12] proposed three different approaches in order to sequentially (incrementally) learn Bayesian Networks. They claim that effective sequential update of structure involves a tradeoff between the quality of the learned network and the amount of information that is maintained about past observations. The three approaches they proposed manage differently the tradeoff. Two of these approaches lie on the extreme of the spectrum, while the third allows for a flexible manipulation of the tradeoff.

On one extreme we have the *naive* approach which stores all previously seen data, and repeatedly invokes a batch learning procedure after each new example is recorded. This approach can use all of the information provided so far, and thus is essentially optimal in terms of the quality of the networks it can induce. This approach, however, requires vast amount of memory to store the entire corpus of data.

On the other extreme we have the second approach, Maximum Aposteriori Probability (MAP) which avoids the overhead of storing all of the previously seen data instances by summarizing them using the model we have seen so far. This approach, similar to that of Lam and Bacchus (see Section 5.2) in the sense that both use one single network structure as a summary of past data, is space efficient. Unfortunately, by using the current model as a summary of past data, we strongly bias the learning procedure towards this model. As a result, after some number of iterations, this approach locks itself into a particular model and stops adapting to new data.

The third approach, which they call *incremental*, provides a middle ground between the extremes defined by the naive and MAP approaches. Moreover, it allows flexible choices in the tradeoff between space and quality of the induced network. The incremental approach interleaves steps in a search process, to find "good" models, with the incorporation of new data. This approach focuses its resources on keeping track of just enough information to make the next decision in the search process. The basic strategy is to maintain a set of network candidates that they call the *frontier* of the search process. As each new data example arrives, the procedure updates the information stored in memory, and invokes the search process to check whether one of the networks in the frontier is deemed more suitable than the current model.

We want to stress that this approach has some similarities with that of Buntine (see Section 5.1) with the open, alive and asleep lists, in the sense that both maintain a set of candidate networks and that both allow a tradeoff space and quality.

5.3.1 Sequential update of Bayesian networks

The naive approach to sequential update consists of storing all the observed data, and then repeatedly invoking a batch learning process. It needs to store either all of the instances that have been observed, or keep a count of number of times each distinct instantiation to all the variables in the dataset was observed. This representation grows linearly with the number of examples observed, and will become infeasible when the network is expected to perform for long periods of time.

The MAP approach is motivated by Bayesian learning methodology. Recall that in Bayesian analysis we start with a prior probability over possible hypotheses (models and their quantifications), and compute the posterior given our observations. In principle, we can then treat this posterior as our prior for the next iteration in the sequential process. Thus, we maintain our *belief state* about the possible hypotheses after observing D_{l-1}. Upon receiving the l-th data example, we compute the posterior as our current belief state. This methodology has the attractive property that in the presence of some reasonable assumptions, the belief state at time l is the same as the posterior of seeing D_l from our initial prior belief state.

If we attempt to use *priors* in order to represent (and update) the posterior we need to store a *complete* network. Unfortunately, this is equivalent to storing the counts for all possible assignments to $\mathbf{X}$.

Since we cannot realize the exact Bayesian network, we can resort to the following approximation. At each step, we find (or approximate) the *maximum a-posteriori probability* (MAP) network candidate. That is, the candidate that is considered most probable given the data so far. We then approximate the posterior in the next iteration by using the MAP network as the prior network. In other words, this procedure uses the network S_l as a summary of the first l observations. This procedure is space efficient since we only need to store the new instances that have been observed since we last performed the update of the MAP.

Unfortunately, by using the MAP model as the prior for the next iteration of learning, we are loosing information, and are strongly biasing the learning process toward the MAP model itself. This phenomena becomes more pronounced as the equivalence sample size is assigned to the prior grows.

In order to overcome the problems found in the former approaches Friedman and Goldszmidt propose a new algorithm they call *incremental*. Unlike the naive approach, it does not keep all possible data examples, and unlike the MAP approach, it does not relay on a single network to represent the prior information. The basic component of this algorithm is a module that maintains a set ST of sufficient statistics records. These records allow the update procedure to select amongst a set of possible networks for the update. Before explaining the approach in detail we introduce some necessary notation. Let *Suff(S)* to denote the set of sufficient statistics for S, that is , *Suff(S)* $= \{N_{i\mathbf{Pa}_i} : 1 \leq i \leq n\}$. Similarly, given a set ST of sufficient statistics records, let $Nets(ST)$ to be the set of network structures that can be evaluated using the records in ST, that is, $Nets(ST) = \{S :$ *Suff(S)* $\subseteq ST\}$.

Suppose that we are deliberating on the choice between two structures S and S'. As we know from the previous sections, in order to use the MDL and Bayesian based measures to evaluate S and S', we need to maintain both the set *Suff(S)* and *Suff(S')*. Now suppose that S and S' differ only by one arc from X_i to X_m. Then there is a large overlap between *Suff(S)* and *Suff(S')*. Namely, *Suff(S)* $\cup$ *Suff(S')* $=$ *Suff(S)* $\cup \{N_{m\mathbf{Pa}_m}\}$. Thus, we can easily keep track of both these structures by maintaining a slightly larger set of statistics.

To see how this generalizes to larger sets that covers a considerable subset of the search space recall that the greedy hill climbing search procedure works by comparing its current candidate S to all its *neighbors*. These neighbors are the networks that are one change away (i.e. arc addition, deletion or reversal) from S. Extending the argument above, we see that we can evaluate the set of neighbors of S, by maintaining a bounded set of sufficient statistics. Note that if ST consists of all the sufficient statistics for S and its neighbors, $Nets(ST)$ contains additional networks, including many networks that add several arcs in distinct families in S. Also note that if $\mathbf{X} \subset \mathbf{Y}$, then $N_{\mathbf{X}}$ can be recovered from $N_{\mathbf{Y}}$. Thus, $Nets(S)$ also contains many networks that are simpler than S.

Generalizing this discussion, this approach applies to any search procedure that can define a *search frontier.* This frontier consists of all the networks it compares in the next iteration. We use F to denote this set of networks. The choice of F determines which sufficient statistics are maintained in memory. That is, we set ST to contain all the sufficient statistics needed to evaluate the networks in F. After a new instance is received (or, in general, after some number of new instances are received), the procedure uses the sufficient statistics in ST to evaluate and select the best scoring network in the frontier F. Once this choice is made, it invokes the search procedure to determine the next frontier, and updates ST accordingly. This process may start recording new information and may also remove some sufficient statistics from memory.

The main loop of the incremental procedure can be now described as in algorithm 11. This procedure focuses its resources on keeping track of just enough information to make the next decision in the search space. Every k steps, the procedure performs this decision. After each such decision is made, the procedure reallocates its resources in preparation for the next iteration. This reallocation may involve removing some sufficient statistics from ST, and adding new ones.

When we instantiate this procedure with the greedy hill climbing procedure, the frontier

Procedure 11 Incremental

Require: initial network S and an initial set frontier F for S

Ensure: (S, θ) an aproximately MAP network

```
1:  $ST = Suff(S) \cup \bigcup_{S' \in F} Suff(S')$
2:  loop
3:     Read data $u_l$
4:     Update each record in $ST$ using $u_l$
5:     if $n \ mod \ k = 0$ then
6:        $S = arg \max_{S' \in Nets(S')}$ Quality$(S'|ST)$
7:        Update the frontier $F$ {using a search procedure}
8:        Set $ST$ to $Suff(S) \cup \bigcup_{S' \in F} Suff(S')$
9:     end if
10:    Compute optimal parameters $\theta$ for $S$ from $ST$
11:    Output$(S, \theta)$
12: end loop
```

consists of all the neighbors of S_l. A beam search, on the other hand, can maintain j candidates, and set the frontier to be all the neighbors of all j candidates. Other search procedures might explore only some of the neighbors of S_l and thus would have smaller search frontiers.

5.3.2 Sequential scoring functions

Friedman and Goldszmidt are the only ones to stress a problem of using the MDL and Bayesian measures in order to compare two models in the sequential learning context. Namely, these two measures assume that we are evaluating all candidates with respect the same dataset.

The underlying problem is a general model selection problem, where we have to compare two models M_1 and M_2 such that model M_1 is evaluated with respect to the training set D_1, while model M_2 is evaluated with respect to the training set D_2. Of course, for this problem to be meaningful, we assume that D_1 and D_2 are both sampled from the same underlying distribution. This assumption is clearly true in our case.

The MDL and the Bayesian scores are inappropriate for this problem in their current form. The MDL score measures the number of bits required to encode the training data if we assume that the underlying distribution has the form specified by the model. However, if D_2 is much smaller than D_1, then the description of D_2 would usually be shorter than that of D_1 regardless of how good the model M_2 is. The same problem occurs with the Bayesian score. This score evaluates the probability of the dataset if we assume that the underlying distribution has the form specified by the model. Again, if D_2 is much smaller than D_1, then the probability associated with it will usually be larger, since th probability of a dataset is a product of the probability of each instance given the previous ones. Since each such term is usually smaller than 1, the probability decreases for longer sequences.

Friedman and Goldszmidt claim that we need a score that assigns higher confidence to families for which we have more data. The reader is referred to [12] for the solution proposed for the MDL score.

5.4 Comments to the incremental proposals

In this section we comment on different incremental learning algorithms so far discussed in the field of Bayesian network learning. We compare the three revised incremental algorithms with Langley's and Friedman and Goldszmidt's definition. We also want to make some references to the problems of the search strategies reported in Section 2.3.2.

We first want to note that the three incremental algorithms follow the definition of incrementality given by Friedman and Goldszmidt in the sense that all of them are able to modify the network structure when new data is available. Even though we think that Buntine's and Friedman and Goldszmidt's proposals are quite similar one each other while Lam and Bacchus' one uses another approximation to incrementality. Both Buntine's and Friedman and Goldszmidt's incremental learning algorithms use the new data items to update the sufficient statistics and thereof to update the posterior probabilities. With these updated probabilities, the algorithms perform additional search over the space of alternative Bayesian networks. As we have seen, the Lam and Bacchus' algorithm use the new data in another way. The algorithm learns a new graph, possibly a subgraph of the old one, and afterwards uses it to update the old one.

Returning to the Friedman and Goldszmidt's proposal, even they do not explicitly make any reference to Langley's definition, they explore from their own proposal the different possible incremental approaches with respect to the restrictions introduced by Langley. Namely, their naive approach which reprocess all data, their MAP approach which keeps one single structure in memory, and finally their incremental approach which processes a reduced set of new data instances in each iteration and keeps in memory a set of significant Bayesian networks. We note that the MAP approach performs a hill climbing search and thus is memory and time efficient. Since it looses information in each iteration of learning, it biases the learning process toward the already learned model.

This problem can be viewed as the ordering problem we introduced in section 2.3.2 where the hill climbing search sticks to a local maxima. Probably, if the MAP algorithm has processed the data in another order, it would have come up with another Bayesian network structure. In order to overcome this problem Friedman and Goldszmidt propose what they call the incremental approach. It keeps in memory more than one Bayesian structure (a frontier) loosing in this way less information and being able to carry out a beam search. Hence, the learning process has more chances to reach a higher maximum.

Buntine also proposes a beam search which keeps several domain models in memory, namely in open, alive and sleep lists. This different domain models give the algorithm the capability of recovering from badly decisions made under a skewed vision of the domain (the database seen so far). However, Buntine provides his algorithm with three parameters in order to control the amount of memory spent in keeping these domain models and time spent performing the search. We want to stress that by means of these parameters the algorithm can be converted into a hill climbing strategy. Still further, we claim that both Buntine's and Friedman and Goldszmidt's beam search proposals may be seen as a generalization of a hill climbing proposal, since hill climbing is a one beam search. We miss in Friedman and Goldszmidt's proposal a parameter in order to state the number of alternative Bayesian networks maintained into the frontier; note that the frontier of their proposal may be seen as the open list of the Buntine's one.

Buntine says that his incremental algorithm rather than processing single new data instances, waits to have a batch of them. Friedman and Goldszmidt give an explicit parameter in order to state the amount of data required in order to proceed to update the network structure. The reason for using granularity is that the evidence acquired with a single new data item may not be enough in order to change ones beliefs.

Going back to the Lam and Bacchus' proposal we would like to note that it is quite different from the other two. They speak, in their article, of refinement rather than of incremental learning, even though, they do not give any precise definition of refinement. However, the fact that the algorithm learns a new graph with the new data instances obligates the algorithm to wait until a great amount of new data is available. Thus, we would not say that Lam and Bacchus' algorithm is incremental in the sense proposed by Langley since, unlike the other incremental algorithms, this one cannot decide to carry out refinement with one or few data instances.

It also seems to us rather arbitrary the way their algorithm modifies the network structure once it has obtained the partial network in the light of the new data. It only reverses an arc when it does not introduce any cycle into the whole network structure, even if the new data clearly states there is a reversed arc. Since reversing such an arc would mean also to change the part of the network structure we do no have any further data, they decided not to reverse the arc.

We also want to comment the problem introduced by Friedman and Goldszmidt. They claim that in the incremental or sequential learning context we compare models evaluated with respect different datasets. It is not clear to us since, in each learning step, the sufficient statistics are updated using the new data items and, only afterwards, the already learnt model is compared to its neighbors. Therefore, they all may be evaluated with respect the updated statistics and thus with respect the same dataset.

As a conclusion we can say that both Buntine's and Friedman and Goldszmidt's proposals follow Langley's definition even they relax its constraints. On the contrary Lam and Bacchus' proposal does not follow the spirit of Langley's definition. We belief that stating a definition of theory refinement, it would give some clues in order to develop further Lam and Bacchus' proposal.

6 Our Proposal

In this section we try to establish what could be our contribution to the Bayesian network learning field. We think that it has been done many interesting work. Even though, many of the people working in this area come from the statistical community and hence, they have mostly developed the numerical fashion of the field. We believe it is also worth developing the field from a Machine Learning viewpoint.

We propose, BANDOLER: BAyesian Network anD On-line LEaRning, a system for learning Bayesian networks from data and prior knowledge. We put the accent, in this system, to the prior knowledge and incremental learning. Even BANDOLER is an unsupervised learning system it may be fed with partial knowledge, obtained from a domain expert, together with a data set. The prior knowledge will be given in a *declarative* way. This sort of knowledge cannot be used by BANDOLER so it must be transformed into a *procedural* one. In our system the procedural knowledge is assumed to be a bias for the searching process.

We would like the domain expert to be able to give prior knowledge during the whole learning process. Thus, BANDOLER should be an incremental or on-line learning system. In this way, BANDOLER will be able to stop learning at any stage of the process, interactuate with the expert and then resume the learning process.

We can see in Figure 3 that the system has two main modules. First, a module to transform declarative into procedural knowledge, and second a module which represents the searching algorithm. We can also identify in Figure 3 different components of the searching module. Namely, a data filter, a set of alternative models, as set of knowledge transmutation operators, a scoring function, and finally a searching strategy. We also see in the picture that each of the components may have a bias, created from the declarative prior knowledge, that is clearly separated and identifiably.

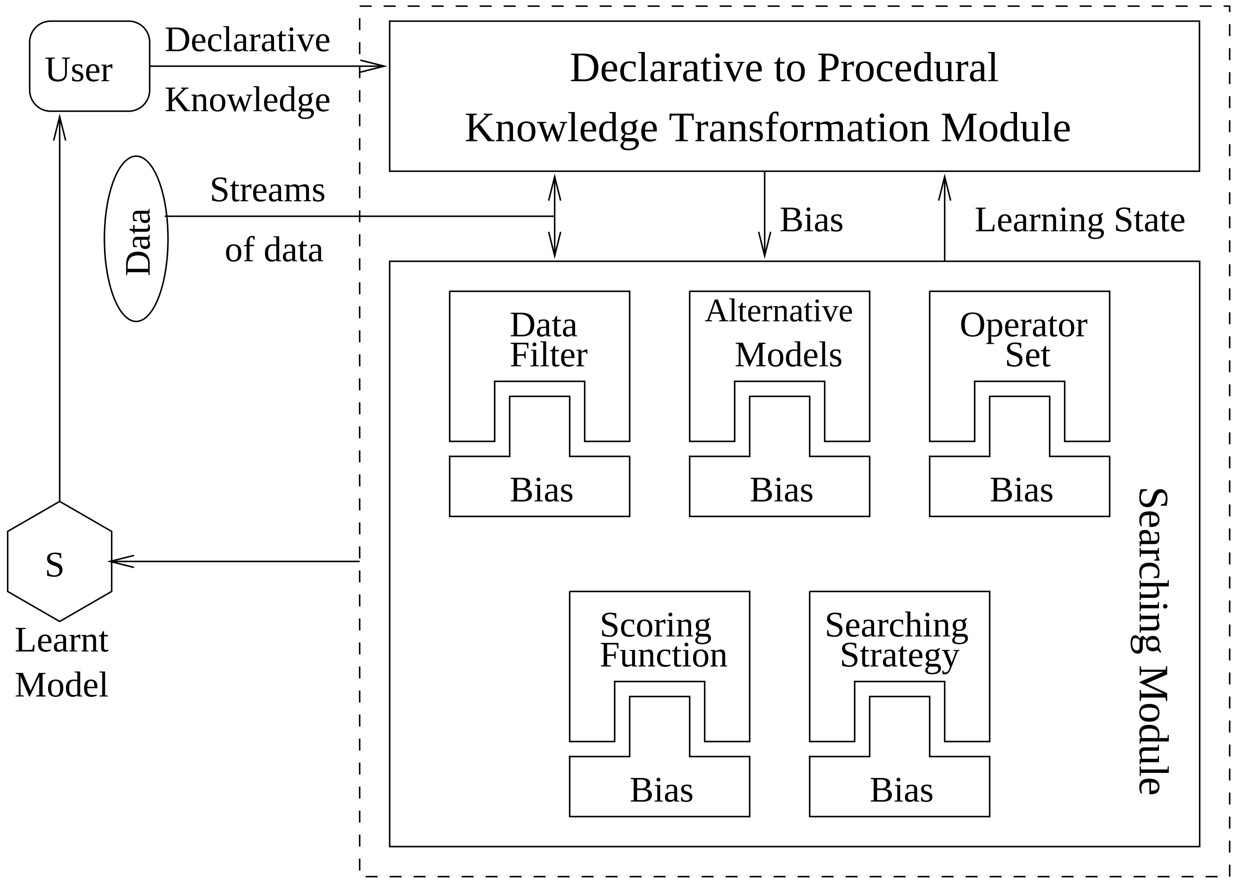

Figure 3: BANDOLER: a system for Bayesian network learning

We belief there is a great amount of research to be done in order to transform the prior knowledge into some kind of procedural knowledge that a learning system may be able to use. We think that the procedural knowledge can be expressed as bias. In this way the prior knowledge helps the learning algorithm to choose the Bayesian network with highest quality score. We also belief it is worth studying the different kinds of prior knowledge, as well as the different kinds of bias a learning algorithm may be fed with.

Background or prior knowledge

In our approach to learning Bayesian networks we assume that we work in an unsupervised environment, that is, algorithms are neither provided with the structure of a network nor with specially selected and relevant data. Even though, there may be domain experts who can provide the learning process with knowledge. The prior knowledge given by an expert will usually be declarative. The expert may feed the system with:

- **A set of selected data instances**: the expert may have an idea of which are the most significant data instances from the data set. If the searching algorithm is incremental, this significant data may be first given to the algorithm in order to obtain an initial Bayesian network.

- **A total or partial order among variables**: this is a way to state the causal precedence between variables.

- **A partial Bayesian network structure**: the expert may know the causal structure of a subset of variables.

- **Causation among pairs of variables**: the expert may give statements such X cause Y, or X does not cause Y.

We also should study what prior knowledge is worth using. For example, it is not useful that an expert states that X causes Y when it can clearly be stated from data. From this point, it seems to us that it could be interesting an interactive algorithm, that is, an algorithm able to demand information from the expert when it lacks into the database.

It may happen that an expert states that X causes Y when from the dataset the algorithm discovers that X does not cause Y. Here we have two contradictory sources of information. We may think of two situation, on the one hand, since we are on a unsupervised context where data is *read* from the domain, it may occur that data is not a good domain sample. Hence, we trust experts, and we would filter those data items that contradict experts.

On the other hand, it may happen that experts have incorrect ideas about the domain and hence give wrong prior knowledge to the algorithm. We should provide some ways (or mechanisms) in order that experts detect this wrong knowledge and hence change their minds and knowledge provided to the algorithm.

Bias

In Section 2 we have identified different points where to apply biases. As we have said prior knowledge may be a source of bias. We will study the bias at different points of the algorithm and at different stages of the learning process:

- **Data filters**: It may be worth doing a study of the dataset before doing the learning process. The results of this study could be used in two ways. First, we may discover that is not worth using the whole dataset (some irrelevant data instances or some irrelevant variables may be thrown away) and second we may obtain some biases to be applied during the searching process. For example, from attribute relevance we may obtain some clue about the causality or the order they could be learnt by an K2-like algorithm.

- **Representation of the Searching space**: if some sort of Bayesian networks are known to be useless beforehand, we could choose a representation of the searching space unable to represent such networks. In this way, the search process will not waste time considering them.

- **Set of operators for knowledge transmutation**: we believe it is worth choosing carefully these operators. They are the basis for exploring the searching space. If the algorithm cannot reach a domain model applying the operators, it would result like not being able to represent such part of the space. Transmutation operators also define the step size of a searching process, which is a critical point for hill climbing algorithms.

- **Evaluation function**: evaluation functions may be a strong bias since they are used to measure the *quality* of the Bayesian networks.

  Evaluation functions for Bayesian learning algorithms have the following general form [3]:

  $$-\log P(B_s) \times -N \times H(B_s, D) - K \times f(n)$$

  where $P(B_s)$ is the a priori probability of the network structure. $H(B_s, D)$ is an expression of the joint cross entropy of the structure and the data, K is a constant factor and $f(n)$ is a *penalty* term that adopts several forms.

  Now it seems clear that when one expresses some prior knowledge each of the terms of this expression for the evaluation functions may be affected. We must note that in the literature usually only the first term, $-\log P(B_s)$, is modified when some prior knowledge is given [16, 5].

  Clearly $P(B_s)$ is affected. If a prior is defined there is an influence of the distribution $P(B_s)$. However, this influence is not restricted to these term. In effect, the formulation of $H(B_s, D)$ is in fact an expression of how to estimate the local probability distributions factorized by the structure B_s. So, given that prior information has been given (for example, order, links list, etc.) then it is clear that not all possible local distribution should be estimated or given the same estimation treatment.

  This transmission of influence form one term to the other has further implications. We are going to study how different types of changes in the a priori information do in fact change such evaluation functions.

- **Search strategy**: the search strategy may consist on choosing a hill climbing rather than a beam search or vice versa, or on choosing the number of beams to be used. However, we belief that choosing the transmutation operator to be used in each step of the searching process may reduce the time required to reach a good Bayesian network.

- **Stages of the learning process**: we belief that biases may change at different moments of the learning process. Probably a bias applied at the beginning of the learning process when the algorithm has still revised few data items is not appropriate at a moment when great amount of data has been revised and the process has much more evidence about the learnt Bayesian network structure.

Incremental algorithms

We have largely argued, see Section 2.3, why is worth developing incremental algorithms. In addition to those arguments we belief that incremental algorithms are useful two reasons. First, for developing interactive systems where the domain expert is able to provide such system with new background knowledge, and second, for changing biases depending on the current stage of the learning process.

In this way, the system may stop the learning process, revise the already learnt Bayesian network structure, accept new prior knowledge or change some biases (in the light of the learnt structure), and then resume the learning process. We also want to note that the learning process may be stopped either by the domain expert (i.e. she revises the already learnt Bayesian network structure and finds out some clue to help the learning process) or by the system itself (i.e. it finds some contradiction with respect the prior knowledge).

We also have revised the proposals in the Bayesian network field. We will study the following points in the context of incremental or on-line learning

- A definition of what is understood by incremental learning of Bayesian networks.
- A definition of what is understood by Bayesian network refinement. Also it should be done a comparison with respect incrementality.
- A deeper exploration of hill climbing and beam search strategies. For example, it is needed to study how many beams should be maintained and in which situations. How should be the beams? Close one to each other, or on the contrary, they should cover different parts of the searching space?
- Develop those algorithms that learn incrementally the variables of a Bayesian network as information about them is acquired. For example, study further the K2 algorithm from this viewpoint.
- Study the conditions for triggering an interactuation with domain experts. For example:
  - The system finds a contradiction between the dataset and the prior knowledge.
  - The system has no evidence enough to choose a Bayesian network structure from a set of candidates.
- Study the use of the already learnt Bayesian network structure as a bias for the following learning steps.
- Study conditions, depending on the learning stage, in order to modify biases.

References

[1] J. R. Anderson and M. Matessa. Explorations of an incremental, bayesian algorithm for categoriz ation. *Machine Learning*, (9):275–308, 1992.

[2] J. Béjar. *Adquisición automática de conocimiento en dominios poco estructurados.* PhD thesis, Facultat d'Informàtica de Barcelona, UPC, 1995.

[3] R.R. Bouckaert. *Bayesian belief networks: from inference to construction.* PhD thesis, Faculteit Wiskunde en Informatica, Utrecht University, 1995.

[4] W. Buntine. Theory refinement on Bayesian networks. In P. Smets B.D. D'Ambrosio and P.P. Bonisone, editors, *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 52–60, 1991.

[5] R. Castelo and A. Siebes. Priors on network structures. Biasing the search for Bayesian networks. In *First International Wokshop on Causal Networks, CANEW-98*, 1998.

[6] E. Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63, 1991.

[7] C.K. Chow and C.N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Teory*, 14:462–467, 1968.

[8] G. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks form data. *Machine Learning*, 9:309–347, 1992.

[9] D. Geiger D. Heckerman and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, (20):197–243, 1995.

[10] L. Xu D.H. Fisher and N. Zard. Ordering effects in clustering. In *Ninth International Conference on Machine Learning*, pages 163–168, 1992.

[11] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, (2):139–172, 1987.

[12] N. Friedman and M. Goldszmidt. Sequential update of Bayesian network structure. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence97*, 1997.

[13] J. H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Artificial Intelligence*, (40):11–61, 1989.

[14] D. F. Gordon and M. Desjardins. Evaluation and selection of biases in machine learning. *Machine Learning*, 20:5–22, 1995.

[15] D. Heckerman. A tutorial on learning Bayesian networks. Technical Report MSR-TR-95-6, Microsoft Research, Advanced Technology Division, 1995.

[16] D. Heckerman and D. Geiger. Likelihoods and parameter priors for Bayesian networks. Technical Report MSR-TR-95-54, Microsoft Research, Advanced Technology Division, 1995.

[17] E.H. Herskovitz and G. Cooper. Kutató: an entropy-driven system for the construction of probabilistic expert systems form data. In *Proceedings of the sixth conference on Uncertainty in Artificial Intelligence*, 1990.

[18] F. V. Jensen. *An introduction to Bayesian Networks.* UCL Press, 1998.

[19] W. Lam and F. Bacchus. Learning Bayesian belief networks. an approach based on the MDL principle. *Computational Intelligence*, 10(4):269–293, 1994.

[20] W. Lam and F. Bacchus. Using new data to refine Bayesian networks. In R. López de Mantaras and D. Poole, editors, *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 383–390, 1994.

[21] P. Langley. Order effects in incremental learning. In P. Reimann and H. Spada, editors, *Learning in humans and machines: Towards an Interdisciplinary Learning Science.* Pergamon, 1995.

[22] M. Lebowitz. Deferred commitment in unimem: waiting to learn. In *Proceedings of the Fifth International Conference on Mac hine Learning*, pages 80–86, 1988.

[23] R. S. Michalski and A. Ram. Learning as goal-driven inference. In Ashwin Ram and David B Leake, editors, *Goal-driven inference*, pages 455–478. The MIT press, 1995.

[24] T. M. Mitchell. The need for biases in learning generalizations. In J. W. Shavlik and T. G. Dietterich, editors, *Readings in Machine Learning*, pages 184–190. 1990. Thechical Report CBM-TR117, Rutgers University.

[25] T. M. Mitchell. *Machine Learning.* Computer Science Series. McGraw-Hill, 1997.

[26] R.E. Neapolitan. *Probabilistic Reasoning in Expert Systems.* Wiley-Interscience, 1990.

[27] J.J. Oliver and D. Hand. Introduction to minimum encoding inference. Technical report, Dep. of Statistics, Open University, 1994.

[28] J.J. Oliver and D. Hand. MML and bayesianism: similarities and differences. Technical report, Dep. of Computer Science, Monash University, 1994.

[29] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[30] T. Rebane and J. Pearl. The recovery of causal poly-trees from statistical data. In T.S. Levitt L.N. Kanal and J.F. Lemmer, editors, *Proceedings of Uncertainy in Atificial Intelligence*, volume 3, North-Holland, Amsterdam, 1989.

[31] J. Roure and L. Talavera. Robust incremental clustering with bad instance orderings: a new strategy. In Helder Coelho, editor, *Progress in Artificial Intelligence–IBERAMIA 98*, Sixth Ibero-American Conference on AI, pages 136–147. Springer, 1998.

[32] R. Sangüesa and U. Cortés. Learning causal networks from data: a survey and a new algorithm to learn possibilistic causal networks from data. *AI Communications*, 19(4):31–61, 1997.

[33] L. Talavera and J. Roure. A buffering strategy to avoid ordering effects in clustering. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of the European Conference on Machine Learning*, pages 316–321. Springer Verlag, 1998.

Notation

$X, Y, Z, \ldots$	Variables or their corresponding nodes in a Bayesian network
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \ldots$	Sets of variables or corresponding sets of nodes
$X = x$	Variable X is in state x
$\mathbf{X} = \mathbf{x}$	The set of variables $\mathbf{X}$ is in configuration $\mathbf{x}$
D	A set of cases
D_l	The first $l - 1$ cases in D
$P(x\|y)$	The probability that $\mathbf{X} = \mathbf{x}$ given $\mathbf{Y} = \mathbf{y}$
$S = (B_S, B_P)$	A Bayesian network with structure B_S (DAG) and B_P the conditional probability assignments associated with the structure
$\mathbf{Pa}_i$	The variable or node corresponding to the parents of node X_i in a Bayesian network structure
p_i	$p_i = \|\mathbf{Pa}_i\|$ the number of parents of node X_i
$\mathbf{pa}_i$	A configuration of the variables $\mathbf{Pa}_i$
r_i	The number of states of discrete variable X_i
q_i	The number of configurations of $\mathbf{Pa}_i$
$\mathbf{S}_i$	A set of the networks with i arcs between variables
S_c	A complete network structure
S^h	The hypotheses corresponding to network structure S
θ_{ijk}	The multinomial parameter corresponding to the probability $P(X_i = x_i^k\|\mathbf{Pa}_i = \mathbf{pa}_i^j)$
θ_{ij}	$= (\theta_{ij2}, \ldots, \theta_{ijr_i})$
θ_i	$= (\theta_{i1}, \ldots, \theta_{iq_i})$
θ_s	$= (\theta_1, \ldots, \theta_n)$
α	An equivalent sample size
α_{ijk}	The Dirichlet hyperparameter corresponding to θijk
α_{ij}	$= \sum_{k=1}^{r_i} \alpha_{ijk}$
N_{ijk}	The number of cases in data set D where $X_i = x_i^k$ and $\mathbf{Pa}_i = \mathbf{pa}_i^j$
N_{ij}	$= \sum_{k=1}^{r_i} N_{ijk}$