UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
TESI DI LAUREA MAGISTRALE

STRUCTURAL LEARNING OF BAYESIAN NETWORKS USING STATISTICAL CONSTRAINTS

RELATORE: Ch.ma Prof.ssa Silvana Badaloni
CORRELATORE: Dr. Francesco Sambo

LAUREANDO: *Francesco Venco*

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Padova, 23 Aprile 2012

Contents

Abstract

Bayesian Networks are probabilistic graphical models that encode in a compact manner the conditional probabilistic relations over a set of random variables. In this thesis we address the NP-complete problem of learning the structure of a Bayesian Network from observed data. We first present two algorithms from the state of the art: the Max Min Parent Children Algorithm (MMPC), Tsamardinos *et al.* [3], which uses statistical tests of independence to restrict the search space for a simple local search algorithm, and a recent complete Branch and Bound technique, de Campos and Ji [5]. We propose in the thesis a novel hybrid algorithm, which uses the constraints given by the MMPC algorithm for reducing the size of the search space of the complete B&B algorithm. Two different statistical tests of independence were implemented: the simple asymptotic test from Tsamardinos et al. [3] and a permutation-based test, more recently proposed by Tsamardinos and Borboudakis [13]. We tested the different techniques for three well known Bayesian Networks in a realistic scenario, with limited memory and data sets with small sample size. Our results are promising and show that the hybrid algorithm exhibits a minimal loss in score, against a considerable gain in computational time, with respect to the original Branch and Bound algorithm, and that none of the two independence tests consistently dominates the other in terms of computational time gain.

Sommario

Le Reti Bayesiane sono modelli grafici probabilistici che cofificano in maniera compatta le relazioni di dipendenza su di un insieme di variabili aleatorie [1]. In questa tesi ci occupiamo del problema NP-Completo di apprendere la struttura di una rete da dei dati osservati. Descriviamo per prima cosa due algoritmi dallo stato dell'arte: l'algoritmo Max Min Parent Children (MMPC) (Tsamardinos et al. [3]) che usa test statistici di indipendenza per restringere lo spazio di ricerca per un semplice algoritmo di ricerca locale, e un algoritmo completo basato su Branch and Bound (de Campos e Ji [5]). In questa tesi proponiamo un nuovo algoritmo ibrido che usa i vincoli creati da MMPC per ridurre lo spazio di ricerca per l'algoritmo completo B&B. Due diversi test sono stati implementati: un semplice test asintotico da Tsamardinos et al. [3] e un test basato su permutazioni più recentemente proposto da Tsamardinos e Borboudakis [13]. Abbiamo sperimentato le diverse tecniche per tre reti ben conosciute in uno scenario realistico, con memoria limitati e scarsità di dati. I nostri risultati sono promettenti e mostrano una minima perdita nella qualità a fronte di un considerevole guadagno in tempo computazionale, e nessuno dei due test ha dominato l'altro in termini di guadagno di tempo.

Chapter 1

Introduction

Since uncertainty appears to be an unavoidable aspect of most real-world applications, many problems in different fields such as medicine, biology and other domains have been tackled with the probabilistic approach of Bayesian Networks. Bayesian Networks are graphical models that represent conditional dependencies over a set of random variables in a compact and intuitive way [1]. It is not difficult to understand why Bayesian Networks have become a popular topic in the last few years.

A Bayesian Network is completely determined by a directed acyclic graph (DAG), whose nodes correspond to the random variables under analysis and whose edges correspond to probabilistic dependencies among the variables, and by the probability distribution of each node.

How to build a Bayesian Network is a challenging task. In some cases, both the underlying structure and the probability distributions can be built with the help of an expert of the domain, whose knowledge may be however incomplete or biased. In many cases experts don't exist at all. The problem we consider is thus the automatic learning of the best Bayesian Network given a set of observed data.

Several scoring functions have been proposed in literature to asses the quality of a Bayesian Network structure: some of the most notable are the *Akaike Information Criterion* (AIC) [6], the *Bayesian Information Criterion* (BIC)[7] and the *Bayesian Dirichlet* (BD) [8].

Given a scoring function, the problem of searching the best BN structure is known to be NP-complete [9] and the number of DAGs with n nodes has been shown to be $O(n!2^{\binom{n}{2}})$ [10]. Exact methods exist but they take time and memory exponential in the number of variables, and are thus limited to small settings. The alternative may consist in the use of heuristic search algorithms, which do not guarantee to find the real optimum. Other approaches, like the PC algorithm [11], rely on statistical tests to find independencies between variables and in a second phase orient the edges to build the DAG. Instead than searching in the space of all possible DAGs, some algorithm explore the space of Bayesian Networks equivalence classes [12].

To overcome all these computational problems, hybrid approaches have also been proposed in recent years. In 2006, Tsamardinos *et al.* proposed a hybrid method [3], Max Min Parent Children (MMPC), which exploits independence tests to retrieve the probable sets of parents and children for every variable in the domain and then performs a heuristic search restricted to these sets. In 2010, Tsmardinos and Borboudakis extended in [13] the MMPC algorithm, providing it with more precise tests for small data set based on random permutation of the data.

In 2008 Perrier *et al.* [14] experimented the use of the 2006 MMPC algorithm in the context of dynamic programming.

Recently, de Campos and Ji presented a new complete algorithm, which can be considered the state-of-the-art in complete search [4][5]. The method is based on a Branch and Bound technique, coupled with a relaxation of the problem and a set of theorems whose application allows to limit the search space. The algorithm is specifically designed to benefit from user-defined constraints on the possible network structures.

In this thesis, we decided to evaluate the benefits and drawbacks of hybridizing the B&B algorithm of de Campos and Ji with both the 2006 [3] and 2010 [13] versions of the MMPC algorithm. Complete algorithms, with enough resources, always find the best solution. On the other hand, heuristic procedures tend to get stuck on not optimum solutions; the MMPC procedure were realized to restrict the search space in an attempt to avoid such local optima [13]. However, we are not assured that, in an non-ideal scenario, the statistical test at the base of MMPC will not exclude the best solution from the search space.

Restricting the search space for a complete algorithm with MMPC can thus provide a predictable gain in time and memory, but the effect on the solution are not trivial. The performance of the two hybrid algorithms is assessed in terms of required computational resources and of quality of the final solution.

We choose for our experiments three well known Networks, already used by Tsamardinos *et al.* in [3], but with much smaller sample sizes and with limitations in the available amount of memory and computational time, to test the hybrid algorithms in a realistic scenario. Experimental results show that, when combined with the B&B algorithm of de Campos and Ji, both versions of the MMPC algorithm result in a substantial gain in computational time, against a limited loss in solution quality. Furthermore, the two hybrid algorithms have smaller memory requirements and are thus able to process larger networks in our limited resources scenario. Interestingly enough, however, none of the two consistently dominates the other.

The remainder of the thesis is organized as follows. We first recall the main definitions of Probability Theory in Chapter 2, along with some key concepts. In Chapter 3 we give the definition of Bayesian Networks and Bayesian Networks most important properties. We introduce also equivalence classes and the related algorithms. In Chapter 4 we introduce Bayesian Network structure learning and explain in detail the topics related to our work. Thus in Chapter 5 we present extensively the most important previous works for our experiments. Finally, in Chapter 6 we fully describe the idea behind our work, the algorithms used and implementations, and we report and discuss our experimental results. In Chapter 7 we present our conclusions, along with some proposals for future work.

Chapter 2

Probability Theory

Before defining formally Bayesian Network, we need to recall a series of concepts and rules of probability theory. This chapter is meant to be just a fast overview on the subject. We will follow for most of definitions and the order of topics the introduction of Bishop's book on machine learning [15].

2.1 Definitions

Probability is the key concept in the field we are considering. Imagine to toss a perfect coin many times: experience tells us that, if the tries are *enough*, the number of tails (or cross) will result to be around 50%. In a similar way, if we roll an ideal balanced dice *many* times, we expect to obtain the face "1" approximately $1/6$ of the tries.

On the other hand, if most of the times we roll a particular dice we obtain a "1", we will suspect the dice has been loaded to cheat games. But how many rolls we should take before we can be certain that we just didn't have bad luck? Intuitively something around ten rolls is not enough; maybe a thousand sounds safer. Moreover, after obtaining a good number of observations, i.e. the fractions of "1" on the total of tries, we will have a rough idea of our chances of obtaining that particular result at every roll. Formally we can define probability in as follows:

Definition 2.1 (Probability)**.** *We define the probability of an event to be the fraction of times that event occurs out of the total number of trials, in the limit that the total number of trials goes to infinity [15].*

We can also define a probability space as follows:

Definition 2.2 (Probability Space). *Given a process or experiment whose results occur randomly, a probability space is a triple* (Ω, F, P) *where* Ω *is the set of all possible outcomes,* F *is a set of events and each event is a set containing zero or more outcomes, and* P *a function from events to probability levels.*

The sentence "rolling a dice we obtain 1" is an event. Events are sets of outcomes in an experiments, and subsets of the sample space Ω, which is the set of all possible outcomes we are considering in a given context. The probability of the sample space is by definition 1, while the probability of the empty event is 0. The probability of all disjoint events in a sample space must sum to 1.

If we call the event of obtaining "1" rolling a dice as $D = 1$, we write the probability of $D = 1$ as:

$$P(D = 1) = 1/6$$

In an ideal dice we have

$$P(D = i) = 1/6 \quad \forall i = 1, ..., 6$$

and in general, no matter how loaded are the faces:

$$\sum_{i=1}^{6} P(D = i) = 1$$

Instead of events we usually prefer to speak of random variables. We recall here the definition [16]

Definition 2.3 (Random Variable). *Let* (Ω, F, P) *be a probability space and* (E, ϵ) *a measurable space. Then an* (E, ϵ)*-valued random variable is a function* $x : \Omega \rightarrow E$ *which is* (F, ϵ)*-measurable .*

Random variable can be *continuous* or *discrete.* In the example of dices, we can define a discrete random variable x which can take the values $\{1, 2, 3, 4, 5, 6\}$, each one corresponding to a face of the dice and with a probability associated to each. Of course the equation $\sum_{i=1}^{6} P(x = i) = 1$ is still valid. In the discrete case, we can define for a variable x a function p_x, called *probability mass function,* that gives the probability that a discrete random variable is exactly equal to some value. In the general case $\sum_{\forall i} p_x(i) = 1$, where i are the possible values for the variable (which can be finite ore not).

Suppose now we have two random variable x and y. The probability that x will take the value a and y will take the value b is written $P(x = a, y = b)$ and is called the *joint probability* of $x = a$ and $y = b$ [15]. We can define the joint probability mass function for two ore more variable, simply following the definition. We will a notation of the kind: $p_{x,y}(a, b)$.

We recall that two variable are said to be *independent* if and only if

$$P(x = a, y = b) = P(x = a)P(y = b) \quad \forall a, b$$

2.2 Fundamental Rules

Let consider now a different situation. Suppose we have 4 balls in a box, two red and two blue. The probability of picking a blue or a red ball from the boxes without looking in the box is equal.

We write as usual $P(x_1 = b) = (x_1 = r) = 0.5$, indicating with x_1 the random variable representing this first extraction from the box, and with b and r the two possible outcomes. Now suppose we extract a second ball from the box, without putting back the first: the probabilities of randomly choosing a red ball are determined by the number of red balls still in the box, i.e. by the color of the first ball extracted. The probability is in fact *conditioned* by the outcome of the first event [15]. We indicate the second extraction with a new random variable, x_2, and we write the conditioned probability of picking a red or blue ball *given the fact* that we first extracted a blue one as follows:

$$P(x_2 = r|x_1 = b) = 2/3$$

$$P(x_2 = b|x_1 = b) = 1/3$$

The calculation is simple: if the first extraction was blue, we have now two red balls and only one blue remaining. In the same way, if we first picked a red ball, and only one red is remaining on the three:

$$P(x_2 = r|x_1 = r) = 1/3$$

$$P(x_2 = b|x_1 = r) = 2/3$$

It's easy to see that the previous four equations define a probability mass function on the variable x_2 conditioned on x_1. We will use in similar cases the notation

$p_{x|y}(a|b)$, where x and y are the considered variables and a and b the value assumed.

So far we have been quite careful to make a distinction between a random variable, such as the die x, and the values that the random variable can assume, for example 1 for the corresponding dice face. Thus the probability that x takes the value 1 is denoted $P(x = 1)$. Although this helps to avoid ambiguity, it leads to a rather cumbersome notation, and in many cases there will be no need for such pedantry.

Instead, following a common praxis, we may simply write $p(x)$ to denote a distribution over the random variable x, or $p(1)$ to denote the distribution evaluated for the particular value 1, provided that the interpretation is clear from the context.

With this more compact notation, we can recall the two well known fundamental rules of probability theory. The first one is called sum rule, or law of total probability [15]

Sum Rule

$$p(x) = \sum_y p(x, y) \tag{2.1}$$

Sometimes all the values for a joint probability mass function can be hard to find, and maybe we know the conditioned probability. In such cases the equation known as product rule [15] can be useful:

Product Rule

$$p(x, y) = p(y|x)p(x) \tag{2.2}$$

From the product rule, together with the symmetry property $p(x, y) = p(y, x)$, we immediately obtain the following relationship between conditional probabilities [15]:

Bayes' Theorem

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \tag{2.3}$$

Bayes' theorem plays a central role in machine learning, and as the name suggests and as we will see later is fundamental for the definition of Bayesian Networks.

Using 2.2, we can also re-write the sum rule as follow:

$$p(x) = \sum_y p(x, y) = \sum_y p(x|y)p(y)$$

We can view the denominator in Bayes' theorem as being the normalization constant required to ensure that the sum of the conditional probability on the left-hand side of 2.3 over all values of y equals one.

2.3 Probability Densities and Distributions

As we mentioned, random variables can be, roughly speaking, discrete or continues. Until now, we only considered in our examples discrete variables, like the ones representing dices or coloured balls. Of course we could be interested in more complex and continues phenomena. We could consider as example the results of clinical analysis, the temperature and humidity in a room, etc. It's important to note that due the digital nature of machine learning we will need to discretize and quantize our data anyway. We recall here some definitions [15] for both continues and discrete variable to set the notation and help the expositions in the following chapters.

In general, we say that a *probability distribution* is a function that describes the probability of a random variable taking certain values. In the discrete domain the probability distribution is characterized by the probability mass function, which is simply a function associating each possible value that can be assumed by the variable with a certain level of probability.

In the continue cases, instead of a mass function we associate to a variable x a probability density function p, where p is a non-negative Lebesgue-integrable function for which:

$$P(x \in (a,b)) = \int_a^b p(x)dx$$

and satisfies the conditions:

$$p(x) \geq 0$$
$$\int_{-\infty}^{+\infty} p(x)dx = 1$$

The distribution F for a continuous variable takes the form of a cumulative distribution function defined by

$$F(z) = \int_{-\infty}^{z} p(x)dx$$

The sum and product rules of probability, as well as Bayes' theorem, apply equally to the case of probability densities, or to combinations of discrete and continuous variables. For instance, if x and y are two real variables, then the sum and product rules take the form:

$$p(x) = \int p(x, y)dy$$
$$p(x, y) = p(y|x)p(x)$$

2.4 Expectations

We recall now the concept of expectations of random variables. We first give the general definition of expectation. Given a continuous random variable x, the expected value of x or expectation of x, denoted by $E[x]$, is defined as Lebesgue integral

$$E[x] = \int_{-\infty}^{+\infty} xp(x)dx \tag{2.4}$$

The average value of some function $g(x)$ under the probability distribution $p(x)$ is called expectation of $g(x)$, will be denoted by $E[g]$ and is given by

$$E[g] = \int_{-\infty}^{+\infty} g(x)p(x)dx \tag{2.5}$$

In the discrete case, the definitions are exactly the same, but we use sums instead of integrals. 2.4 and 2.5 become then

$$E[x] = \sum_x xp(x) \tag{2.6}$$
$$E[g] = \sum_x g(x)p(x) \tag{2.7}$$

The expectation $E[x]$ is also called *mean* of x and is often indicated with μ, while $\sqrt{E[(x-\mu)^2]} = \sigma$ is called *standard deviation*, σ^2 is called *variance.*

2.5 Important Distributions

We recall here some important distributions that will be used in the following chapters

2.5.1 Normal Distribution

The *Normal or Gaussian* distribution is a continuous probability distribution that has a bell-shaped probability density function, known as the Gaussian function or informally the bell curve:

$$p(x, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where parameter μ is the mean and σ^2 is the variance. The distribution with $\mu = 0$ and $\sigma^2 = 1$ is called the standard normal distribution or the unit normal distribution.

2.5.2 χ^2 Distribution

The χ^2_{df} or *chi-squared distribution* with k degrees of freedom is the distribution of a sum of the squares of k independent standard normal random variables. Formally, if $x_1, x_2, ..., x_k$ are k independent, standard normal random variables, then

$$Q = \sum_{i=1}^{k} x_i^2 \sim \chi^2_{df}(k)$$

The cumulative distribution function for $\chi^2_{df}(k)$ can be written as:

$$F(x, k) = \frac{\gamma(\frac{k}{2}, \frac{x}{2})}{\Gamma(\frac{k}{2})}$$

where $\Gamma(z)$ is the gamma function and $\gamma(z, w)$ is the lower incomplete Gamma function.

2.5.3 Dirichlet Distribution

The *Dirichlet distribution* (after Johann Peter Gustav Lejeune Dirichlet), is a family of continuous multivariate probability distributions parametrized by a vector α of positive reals. The Dirichlet Distribution of order K ≥ 2 with parameters $\alpha_1, \alpha_1, ..., \alpha_k > 0$ has a probability density function given by

$$p(x_1, x_2, ..., x_k, \alpha_1, \alpha_2, ..., \alpha_k) = \frac{1}{B(\alpha)} \prod_{i=1}^{k} x_i^{\alpha_i - 1}$$

where $B(\alpha)$ is a normalization factor, $x_1, ..., x_k > 0$ and $x_1 + x_2 + ... + x_k < 1$.

2.5.4 Multinomial Distribution

The Multinomial Distribution models the outcomes from an experiment that consists of n statistically independent repeated trials, each one with a discrete number of possible outcomes x_i. On any given trial, the probability p_i that a particular outcome will occur is constant.

2.6 Bayesian Probabilities

Recalling the example of coloured balls in a box, calculating the probability of extracting a blue or a rad ball is straightforward, and is based uniquely on the number of favourable events (picking one of the two red, no matter which one) on the total number of possibilities.

This assuming that all the balls are indistinguishable at touch and there is no greater possibility of picking a particular one . If we have a dice, for example, and we suspect that it is not a fair one, i.e. one of the faces has been loaded to increase the probability, we can simply roll it many times and use the frequencies of each face to determine an approximation of the distribution for that particular dice. Of course, the more we repeat the experiments, the better our approximation will be.

So far we have viewed probabilities in terms of the frequencies of random, repeatable events. We shall refer to this as the classical or frequentist interpretation of probability [15]. Probability on the other hand can be used in a different way, as a measure of *uncertainty*. Consider an uncertain event , for example whether the moon was once in its own orbit around the sun, or whether the Arctic ice cap will have disappeared by the end of the century. These are events that can't be repeated numerous times in order to define a notion of probability as we can do for a simple dice . In some circumstances we can have however an idea, or some indication.

"If we now obtain fresh evidence, for instance from a new Earth observation satellite gathering novel forms of diagnostic information, we may revise our opinion on the rate of ice loss. Our assessment of such matters will affect the actions we take, for instance the extent to which we endeavour to reduce the emission of greenhouse gasses. In such circumstances, we would like to be able to quantify our expression of uncertainty and make precise revisions of uncertainty in the light of new evidence, as well as subsequently to be able to take optimal actions or decisions as a consequence. This can all be achieved through the elegant, and very general, Bayesian interpretation of probability" [15].

To understand how, we need to return on the box with coloured balls. Suppose that at the beginning there are three blue balls and only two red. We know that the probability of getting a red ball at first extraction is simply $2/5 = 0.4$. If we call x_1 and x_2 the variables associated to the first and second extraction respectively, we can also easily calculate the probability $P(x_1 = r, x_2 = r)$ of tow consecutive red extraction In fact using the product rule:

$$P(x_1 = r, x_2 = r) = P(x_2 = r|x_1 = r)P(x_1 = r) = \frac{1}{4} * \frac{2}{5} = 0.1$$

where again we found $P(x_2 = r|x_1 = r) = 1/4$ simply considering that after one red ball has been extracted, only one remain on the total of four.

What now if we are asked to find $P(x_1 = r|x_2 = r)$, namely the probability of having picked at the first try a red ball, knowing that also the the second extraction was red? First we find $P(x_2 = r)$ using the sum rule:

$$\begin{aligned} P(x_2 = r) &= P(x_2 = r|x_1 = r)P(x_1 = r) + P(x_2 = r|x_1 = b)P(x_1 = b) \\ &= 1/4 * 2/5 + 2/4 * 3/5 = 0.4 \end{aligned}$$

Now we can use the Bayes Theorem to easily give the solution:

$$P(x_1 = r|x_2 = r) = \frac{P(x_2 = r|x_1 = r)P(x_1 = r)}{P(x_2 = r)} = \frac{1/4 * 2/5}{1/4} = 0.4$$

We give that simple example to introduce a very important terminology. If we had asked which ball had been chosen at the first pick, before being told the colour of the second extraction, then the most complete information we have available is provided by the probability $P(x_1 = r)$. That's what's called **prior probability** because it is the probability available before we observe the second extraction. Once we are told that also the second ball was red, we can use the theorem to compute the probability $P(x_1 = r|x_2 = r)$, which we call **posterior probability**, because it is the probability obtained after we have observed the second extraction.

The identical approach can be used for much more complicated cases and in the context of the *Bayesian approach to probability*, i.e. using probability to quantify uncertainty.

Imagine for example we have collected a certain number of coordinates in the space, and we try to recover a model for a supposed underling function. Or maybe we want to automatically build a classifier to distinguish two classes of objects with a certain number of attributes. In every case, we can capture our assumption about the model M, before observing the data, in the form of a prior probability $P(M)$. Using again the bayes theorem, we can evaluate a new and much more significant state of uncertainty on the model given the contribution of the data D:

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)} \tag{2.8}$$

We have already observed how the denominator can be seen just as a normalizing factor. The quantity $P(D|M)$ on the right-side of the equation will have much more importance. It can be seen as a function of the model and we call it **likelihood function**. It expresses how probable the observed data are for different settings of the parameters of the model M.

Given all these definitions and considerations, we can state the theorem in words [15]

$$posterior \propto likelihood \, \text{x} \, prior \tag{2.9}$$

As we will see in details for Bayesian Network, what we want to obtain in many cases is a model which is the best model fitting some observed data. In order to do that, we can try to maximize the posterior probability. Finding a way to express $P(M|D)$ can be troublesome. An easier way is exploit 2.9 and try to maximize the right side of the equation instead.

Giving a mathematical form to a prior probability is sometimes considered a stretch [15], by consequence often it will be simply considered equal for all possible models. In this case we have what is called a non-informative prior and the problem reduces at finding the parameters that maximize only the likelihood function.

We will explain in successive chapter how to mathematically express all the elements specifically in the domain of Bayesian Network; specifically, the model we want to learn is the structure of the network, which represents the relations of conditional dependence between a set of random variables, while the observed data consist in a certain number of entries, each one containing a value for each variable in the problem.

In the next chapters we will gradually explain the problem and the solutions proposed in literature. However, it should be intuitive why finding an expression for the likelihood is normally easier than trying the same operation directly on the posterior.

Chapter 3

Bayesian Networks

3.1 Definition

Bayesian Networks are probabilistic graphical models used to represent in a compact manner probabilistic relations between random variables [1]. It will become apparent that they are very convenient representations for both human users comprehension and mathematical proprieties.

Before we further extend the definition, we recall some notions about graphs. A Directed Acyclic Graph, or DAG, is a couple (V_G, E_G) , where V_G is a set of *vertices* or *nodes* and E_G is a set of *edges* or *arcs* connecting the edges: E_G can be considered as a subset of V_GxV_G. Being the graph directed, we will represent an edge between two nodes $v_1, v_2 \in V_G$ with an arrow: $v_1 \rightarrow v_2$; v_1 is said to be a *parent* of v_2, meanwhile v_2 is a *child* of v_1.

We call a path on the graph a sequence of vertices such that from each of its vertices there is an edge to the next vertex in the sequence. A node v_1 is called *ancestors* of a node v_2 if there exists a path between v_1 and v_2, starting at v_1. Similarly v_2 is a *descendant* of v_1.

Acyclic means that cycles in a DAG are not possible: a cycle is intuitively defined as a path between a node and itself. Two nodes in a graph are said to *adjacent* if there is an arc connecting them, and a set of nodes for wich every pair of nodes is adjacent is called *clique*. We will denote the set of adjacent nodes of a node x, also called neighbours of x, with N_x.

Formally a Bayesian Network can thus be defined as follows [17]:

Definition 3.1 (Bayesian Network). *A Bayesian Network B is a triple* (G, X, Ξ)*, where*

- *G is a Directed Acyclic Graph*
- *X is a (finite) set of Random Variables*
- *Ξ is a collection of Conditional Mass Functions*

In a Bayesian Network the number of nodes of the graph G, also called *structure* of the network, is equal to the number of random variables in X, or in symbols $|V_G| = |X|$, and every node of G corresponds to a different variable in X. In other words every node in the structure of the network represents graphically one of the random variables belonging to X; for this reason we will either refer to nodes and variables unless the context does not require otherwise. For simplicity, we will consider only discrete random variables.

The third element of a Bayesian Network is a collection Ξ of conditional mass functions, one for each variable $x_i \in X$, which take the form $p(x_i|\Pi_i)$, where Π_i is the set of parents of the variable in the graph G. As it naturally follows from these definitions, Bayesian Networks respect the following propriety, called Markov Condition [17]:

Definition 3.2 (Markov Condition). *Any node (or variable) in a Bayesian Network B is conditionally independent of its non-descendants, given his parents.*

In summary, a Bayesian Network represents a joint probability distribution over a collection of n random variables; we can explicit that distribution using the following simple passages [8]. First, given a topological order of the variables, we use the chain rule of probability to write:

$$p(x_1, ..., x_n) = \prod_{i=1}^{n} p(x_i|x_1, ..., x_{i-1}) \tag{3.1}$$

Given the topological order of the variables we can affirm that $\Pi_i \subseteq \{x_1, ..., x_{i-1}\}$, and applying Markov Condition we now that Π_i renders x_i and $\{x_1, ..., x_{i-1}\}$ conditionally independent. That is,

$$p(x_i|x_1, ..., x_{i-1}) = p(x_i|\Pi_i) \tag{3.2}$$

Finally combining 3.1 and 3.2 we obtain

$$p(x_1, ..., x_n) = \prod_{i=1}^{n} p(x_i|\Pi_i) \tag{3.3}$$

It's important to stress that the conditional independences observed in the distribution of a network are not accidental properties of the distribution, but instead due to the structure of the network. This is asserted by the *faithfulness* condition below [18][19][20]:

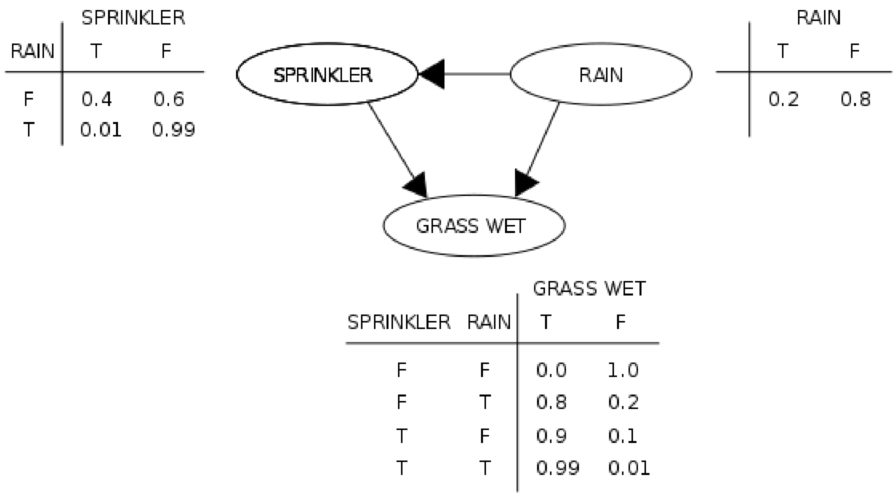
Definition 3.3 (Faithfulness)**.** *If all and only the conditional independences true in the distribution p are entailed by the Markov condition applied to G, we will say that p and G are faithful to each other. Furthermore, a distribution p is faithful if there exists a graph, G, to which it is faithful.*

To fully understand the definitions we make a simple example.

Example 3.1. We consider a set of three discrete random variables. The variables are R "RAIN", S "SPRINKLER" and G "GRASS WET" ; R can be either "true" if today it rained or "false" if not; similarly S is "true" if the sprinkler has been activated, and G is "true" if the grass is wet. Intuitively, if today it rains, probably the sprinkler will not be activated, while the event "the grass is wet" depends on both the rain and the artificial irrigation. We can imagine a Bayesian Network coding the relations between the variables (figure 3.1)

Each one of the three variables is assigned in fig. 3.1 to a node, and the edges are built respecting the relations we explained in words. The tables near each node contains the mass probability functions for the corresponding variable conditioned by its parents.

We can note that the node "RAIN" does not have any parents, thus the table contains only the two probabilities relative to the only two possible states for the variable ("true" or "false"); being the structure of a Bayesian Network acyclic at least one node must be without parents. "SPRINKLER" has only one parent and the relative probability table contains four elements, while "GRASS WET", having two parents, has 8 different probabilities. Of course every line in the tables must sum to 1. We can derive the joint probability distribution vary easily applying 3.3:



	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99

RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Figure 3.1

$$p(R, S, G) = p(R)p(S|R)p(G|R, S)$$

3.2 Applications

Given the definition it should be clear how Bayesian Networks can be an useful toll to model phenomena in the dominion of uncertainty. We give now, before we proceed further, a fast overview on the practical use of Bayesian Networks . More informations can be found in [2] [21] [22] .

Part of the popularity of Bayesian Networks, as we already pointed out, must stem from their visual appeal, as it makes them amenable to analysis and modification by experts. However, as a Bayesian network is a joint probability distribution, any question that can be posed in a probabilistic form can be answered correctly and with a level of confidence. Some examples of these questions are:

- Given some effects, what were the causes?
- How should something be controlled given current readings?
- In the case of causal Bayesian networks, what will happen if an intervention is made on a system?

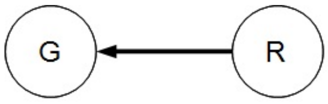
Many of the original applications were in the *medical field* and to some extent, this is the domain where Bayesian network applications dominate today. In medicine, Bayesian networks have been primly used to build system for diagnosis [23] [24]; a very important network is the ALARM network developed by Beinlich *et al.* [25], which was used for the monitoring of patients in intensive care situations. It is often treated as a gold-standard network, as it is reasonably well connected and has enough nodes to be a challenging, but still achievable problem for many Bayesian network algorithms that we will study in next chapters.

Bayesian networks can be very useful in predicting the future based on current knowledge; in other words a domain of application is *forecasting* and *classification* (some examples in [26] and [27]). Some applications have been attempted also in the field of *control* as in [28]. It is also important the use for *modelling for human understanding*, for example in the field of biology [29].

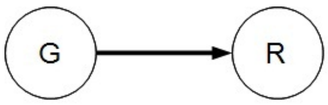
3.3 Equivalence

As we will see, equivalence between network structures has an important role in the automatic learning of Bayesian Network

Suppose we have the following network, a simplified version of the example in figure 3.1:



Then consider the following network:



What is the difference between the two?

Obviously, from a strictly cause/effect point of view, building a network that models the distribution of the two variables, defining the probability of the event "it rained" given that "the grass is wet" may not be the first choice. However, the information given in both cases is exactly the same.

We can prove it easily; first we write the joint probability distribution:

$$p(G, R) = p(R)p(G|R)$$

Now, we apply Bayes Theorem

$$p(G|R) = \frac{p(G)p(R|G)}{p(R)}$$

We then replace $p(G|R)$ in the previous equation and we obtain

$$p(G,R) = p(R)\frac{p(G)p(R|G)}{p(R)} = p(G)p(R|G)$$

which is exactly the joint probability distribution we would have directly written for the second network. The two network structures represent the same relations between the variables, and that can be naturally used as a definition of equivalence. Formally [12]

Definition 3.4 (Equivalence)**.** *Two DAGs G and G' are equivalent if for every Bayesian Network $B = (G, X, \Xi)$, there exists a Bayesian Network $B' = (G', X, \Xi')$ such that B and B' define the same probability distribution, and vice versa.*

In other words, two structures are equivalent if every probability distribution encoded by one structure can be encoded by the other, and vice versa [8], which is exactly the process we followed at the beginning of this section. We examine now an other example, a bit more complex, in order to be able to do some further consideration.

Example 3.2. We consider again a set of three discrete random variables, x_1, x_2 and x_3. We start with the following structure:

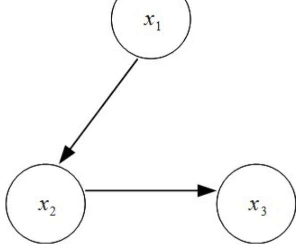


Figure 3.2

we then write the joint probability distribution and apply Bayes theorem to the part relative to the arc $x_1 \to x_2$:

$$\begin{aligned} p(x_1, x_2, x_3) &= p(x_1)\boldsymbol{p(x_2|x_1)}p(x_3|x2) \\ &= p(x_1)\frac{\boldsymbol{p(x_1|x_2)p(x_2)}}{\boldsymbol{p(x_1)}}p(x_3|x2) \\ &= p(x_2)p(x_1|x_2)p(x_3|x2) \end{aligned}$$

Observing the last equation we can see that what we have done is nothing more than *reverting* the arc. As a matter of fact the we could write exactly the same distribution observing the following structure:

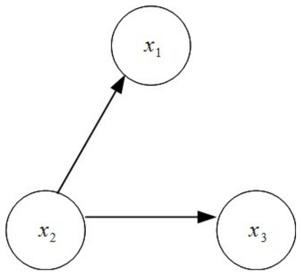


Figure 3.3

Now, we can continue applying again the theorem to the last arc:

$$\begin{aligned} &= p(x_2)p(x_1|x2)\boldsymbol{p(x_3|x_2)} \\ &= p(x_2)p(x_1|x2)\frac{\boldsymbol{p(x_2|x_3)p(x_3)}}{\boldsymbol{p(x_2)}} \\ &= p(x_3)p(x_1|x2)p(x_2|x_3) \end{aligned}$$

Again we obtained a distribution that can be represented by a new graph

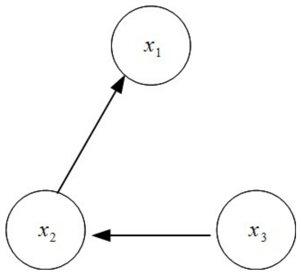


Figure 3.4

Considering that all the equations we wrote are equals it is easy to verify that graphs in figures 3.2, 3.3 and 3.4 are all equivalent to each others simply reading the definition 3.4. But what can we say about the following network?

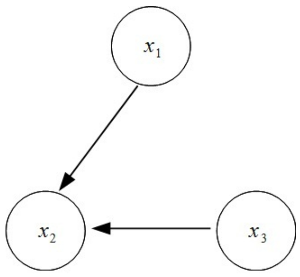


Figure 3.5

We can try to proceed exactly like we did in the first passage, trying to revert $x_2 \rightarrow x_3$ in 3.2 this time:

$$\begin{aligned} p(x_1, x_2, x_3) &= p(x_1)p(x_2|x_1)\boldsymbol{p(x_3|x_2)} \\ &= p(x_1)p(x_2|x1)\frac{\boldsymbol{p(x_2|x_3)p(x_3)}}{\boldsymbol{p(x_2)}} \\ &= \frac{p(x_1)p(x_3)}{p(x_2)}p(x_2|x_1)p(x_2|x_3) \end{aligned}$$

Unfortunately, the distribution we desired is $p(x_1)p(x_3)p(x_2|x_1, x_3)$, which is generally different from the last equation we obtained. Problem is, at this stage we can't assume for sure that 3.5 is not equivalent to the precedent graphs, because we can't be sure we are not missing some mathematical passage that would be able to give us the equation are looking for ...

To better formalize the concept of equivalent networks, and specifically the process we have followed in example 3.2, we recall a theorem from [8]:

Theorem 3.1 (Chickering, 1995)**.** *Let G_1 and G_2 be two Bayesian Network structures, and R be the set of edges by witch G_1 and G_2 differ in directionality. Then, G_1 and G_2 are equivalent if and only if there exists a sequence of $|R|$ distinct arc reversals applied to G_1 with the following proprieties:*

1. *After each reversal, the resulting network structure contains no directed cycles and is equivalent to G_1*

2. *After all reversals, the resulting network structure is identical to G_2*

3. *If $x \to y$ is the next arc to be reversed in the current network structure, then x and y have the same parents in both network structures, with the exception that x is also a parent of y in G_1*

We can note that we followed the rules in our example, but they are not really useful. More importantly, checking the equivalence at every passage can be quite impractical for large networks. There exists a far easier way to check equivalence, but before we need to give a couple of new definitions.

First, we give the definition of the skeleton of a network structure:

Definition 3.5 (Skeleton)**.** *Given a DAG $G = (E, V)$, his skeleton is the undirected graph $G' = (E', V)$ where the set of vertices V is the same in both G and G', and E' is obtained taking every edge from E and ignoring the directionality.*

The other element we need is a particular structure formed by three variable in a graph and called V-structure:

Definition 3.6 (V-structure)**.** *Given a DAG $G = (E, V)$, three nodes $x, y, z \in V$ form a v-structure if x and y are not adjacent and E contains the arcs $x \to z$ and $y \to z$.*

We can now enunciate a simple characterization of equivalent structures derived by Verma and Pearl [30]

Theorem 3.2 (Verma, Pearl 1990)**.** *Two DAGs are equivalent if and only if they have the same skeletons and the same v-structure.*

We can immediately apply the theorem 3.2 to example 3.2: all the networks there have the same skeleton, but the graph in figure 3.5 is exactly made by a single v-stucture which is not in the others structures, thus the last network is not equivalent to the first three. We consider an other case, just a bit more complex

Example 3.3. We start considering the following DAG (a); his skeleton (b) is a simple graph built taking every node of (a) and adding an undirected edge at the position of every directed arc on (a)

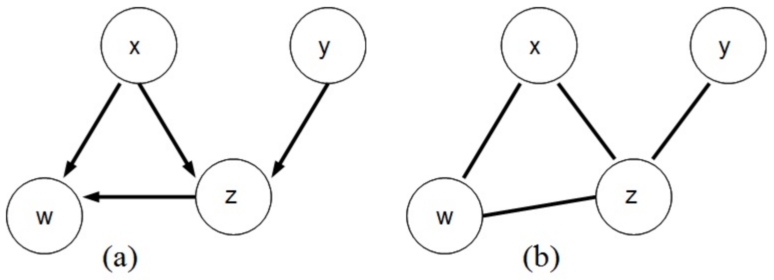


Figure 3.6: A network (a) and his skeleton (b)

We notice a v-structure in graph (a) formed by the arcs $x \to z$ and $y \to z$. It is important to stress that the triple (x, z, w) does not form a v-structure: in fact the node x and z are adjacent. The following networks are built using the same skeleton (b) directing the arcs in different ways, being only careful of not adding cycles:

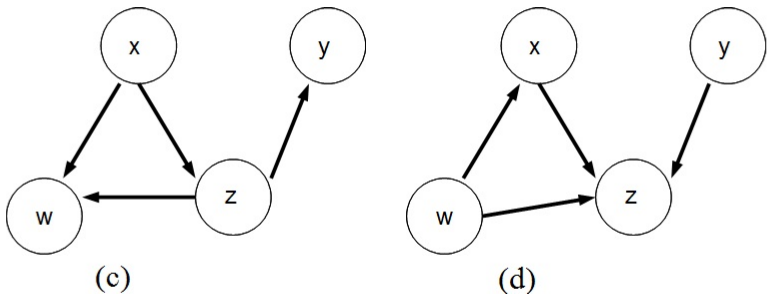


Figure 3.7: Two networks with identical skeleton but different v-structure

The graph in figure (c) differs from the original graph only for the direction of a single edge but does not have any v-structure, while structure (d) has two arcs inverted but presents the same v-structure (x, y, x) we found in (a). It's immediate to conclude that (a) end (d) are equivalent, (d) and (a) are not.

3.4 Equivalence Classes - PDAGs

In some cases the number of Bayesian Network in a domain being equivalent to each others is large, thus it can be useful to do not consider the classical representation for structures, given by DAGs, and try to use some form of Equivalent Class.

First, we will say that a directed edge $x \to y$ is *compelled* in G if for every DAG G' equivalent to G, $x \to y$ exists in G'. For any edge e in G, if e is not compelled in G, then e is *reversible* in G; that is, there exists some DAG G' equivalent to G in which e has opposite orientation.

To represent classes, it has been proposed [12] [31] the use of Partially Directed Acyclic Graphs, in short PDAGs. A PDAG is a graph that contains both directed and undirected edges. It is acyclic in the sense that it contains no directed cycles, while the definitions of clique, adjacent nodes, v-structure and skeleton are in practice identical to those already given for the DAGs. For equivalence classes we will use the following definition

Definition 3.7 (Equivalence Class)**.** *Given a PDAG P and a DAG G, we will define the class represented by P as $Class(P)$ and we will say that $G \in Class(P)$ if and only if G and P have the same skeleton and the same set of v-structures.*

From Theorem 3.2 it follows that a PDAG must contain a directed edge for every edge forming a v-structure, and an undirected edge for every other edge, uniquely identifies an equivalence class. In truth there may be many PDAGs corresponding to the same class: a DAG itself can be seen as a PDAG, and can be taken to represent his own class.

To avoid this ambiguity, we need to pick *completed* PDAGs, or CPDAGs in short, to represent classes [12]. A completed PDAG is a PDAG consisting of a directed edge for every compelled edge in the equivalent class, and an undirected edge for every reversible edge. CPDAGs have an interesting propriety: in fact not every PDAG represents a class of DAGs.

If a DAG G has the same skeleton and the same set of v-structure as a PDAG P, and if every directed edge in P has the same orientation in G, we say that G is a *consistent extension* of P: PDAGs without consistent extension by definition does not have any DAG G in their class [12]. CPDAG on the other hand always have at least one consistent extension; more importantly, from theorem 3.2 if follows that given an equivalent class of Bayesian Network structures, the CPDAG for that class is unique [12].

Example 3.4. Here we have a DAG and the corresponding CPDAG. It's easy to see that in this case only two DAGs belong to the same equivalence class.

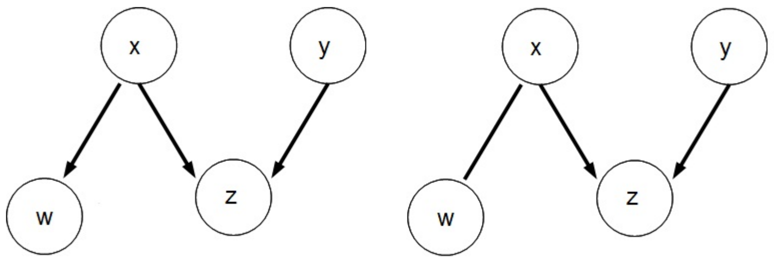


Figure 3.8: A structure (left) and the CPDAG representing its equivalence class (right)

Knowing the definition is not enough to pass from a DAG to the corresponding CPDAG. We describe in the following sections two algorithms. The first is used to pass from an acyclic graph to its equivalence class representative, and it will be called DAG-TO-CPDAG. Vice versa the second build from a PDAG a consistent extension, if exists, and it will be called PDAG-to-DAG.

3.4.1 DAG-to-CPDAG

Different algorithms have been described to build from a DAG the corresponding CPDAG; Verma and Pearl in 1992 and Meek in 1995 [32], presented rule-based algorithms that can be used to implement DAG-to-CPDAG.

The idea of these implementations is as follows. First, we undirect every edge in a DAG, except for those edges that participate in a v-structure. Then, we repeatedly apply one of a set of rules that transform undirected edges into directed edges.

The rules "fire" by matching local patterns of undirected and directed edges in the given graph. Meek [32] proves that the transformation rules are sound and complete. That is, once no rule matches on the current graph, that graph must be a completed PDAG. Andersson *et al.* in 1997 [33] provided a similar rule-based algorithm, except that edges from a DAG are undirected by matching patterns of edges.

Here instead we quote from [12] an algorithm computationally efficient first provided by Chickering in [34], which when given a DAG with $|E|$ edges runs in time $O(|E|)$. The algorithm label all the edges in the DAG as either compelled or reversible. Given the labelling, following the definition building the CPDAG is immediate. We present all the pseudo-code of the algorithm in the following pages. To facilitating the reading every step of the procedure is presented separately.

The first operation needed will be a topological sort of nodes, which we present the pseudo-code in algorithm 1. The second step, algorithm 2, will be to order the edges in the graph. The pseudo-code to label the edges as compelled or reversible is then the following presented in algoritthm 5. The pseudo-code for the DAG-to-CPDAG procedure is obvious and is given at the end for completeness.

```
Algorithm 1: Order Nodes
Data: DAG G
Result: nodes of G in topological order
L = Empty list (it will contain the sorted elements)
S = Set of all nodes with no incoming edges
while S is not empty do
    remove a node n from S
    insert n into L
    for node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S
        end
    end
end
if graph has edges then
    return error (graph has at least one cycle)
end
else
    return L (a topologically sorted order)
end
```

```
Algorithm 2: Order Edges
Data: DAG G
Result: an order for edges of G
use ORDER-NODES on G
Set $i = 0$
while there are unordered edges in G do
    $y$ = lowest ordered NODE that has an unordered EDGE incident to it
    $x$ = highest ordered NODE for which $x \rightarrow y$ is not ordered
    label $x \rightarrow y$ with order $i$
    $i = i + 1$
end
```

```
Algorithm 3: Label Edges
Data: DAG G
Result: edges of G labelled as "compelled" or "reversible"
use ORDER-EDGES on G
label every edge of G as "unknown"
while there are "unknown" edges in G do
    $x \rightarrow y$ = lowest ordered unknown edge
    for every edge $w \rightarrow x$ labelled "compelled" do
        if $w$ is not a parent of $y$ then
            Label $x \rightarrow y$ and every edge incident into $y$ with "compelled"
            continue while
        end
        else
            Label $w \rightarrow y$ "compelled"
        end
    end
    if there exists $z \rightarrow y$ such that $z \neq y$ and $z$ is not a parent of $x$ then
        Label $x \rightarrow y$ and all "unknown" edges incident into $y$ with "compelled"
    end
    else
        Label $x \rightarrow y$ and all "unknown" edges incident into $y$ with "reversible"
    end
end
```

Algorithm 4: DAG-TO-CPDAG

```
Data: DAG G
Result: complete PDAG C representing G
C = empty graph with same nodes than G
use LABEL-EDGES on G
for each edge $x \to y$ in G do
    if $x \to y$ is "compelled" then
        Add $x \to y$ in P
    end
    else
        Add $x - y$ in P
    end
end
return C
```

3.4.2 PDAG-to-DAG

The algorithm we consider for the inverse operation is due to Dor and Tarsi [35]. The procedure is used to build from a PDAG a consistent extension. We recall that there can be many consistent extensions for a PDAG, each one belonging to the same equivalence class. Some PDAG however don't have any consistent extension, while Complete PDAGs always have one.

The algorithm is general and can be applied to any PDAG. If the PDAG is not representative of any directed acyclic graph, the algorithm will find it. We recall that we use N_x to denote the neighbours of x and Π_x the parents of x.

```
Algorithm 5: PDAG-TO-DAG
Data: PDAG P
Result: a consistent extension G of P
G = empty graph with same nodes than P
for each directed edge $y \to x$ in P do
    Add $y \to x$ in G
end
while there are nodes in P do
    select $x$ such that
    - $x$ has not outgoing edges
    - if $N_x$ is not empty, $N_x \cup \Pi_x$ is a clique
    if no such node is found then
        return error (P has not a consistent extension)
    end
    for each undirected edge $y - x$ in P do
        insert $y \to x$ in G
    end
    remove $x$ and all incident edges from P
end
return G
```

At every step a variable without "children", i.e. without outgoing directed edges, is selected. The variable must also meet the condition that if there exist any adjacent node, the set $N_x \cup \Pi_x$ must be a clique. If at any point of the algorithm such x does not exist, the PDAG doesn't admit a consistent extension.

Chickering in [12] give a loose upper bound on the complexity of a simple implementation of the algorithm. If we give in input a PDAG with |V| nodes and |E| edges, for each node x, we can determine if its neighbours and parents form a clique in $O(|N_x \cup \Pi_x|^2)$ time. For each edge that is removed, we need only check the endpoints to see if their neighbours and parents now form a clique; if $N_x \cup \Pi_x$ is a clique for some node z in P at any step of the algorithm, this set must remain a clique after each edge removal in the algorithm. If $|N_x \cup \Pi_x|$ is bounded above by some constant k then we can implement the algorithm in time $O(|V|k^2 + |E|k^2)$.

Of course this upper bound may not be good if k is near or in the order of the number of nodes in the network. However Chickering observes that in practice the graphs used are reasonably sparse.

3.5 Clarification on the notation

We use lower-case letters to indicate both variables and nodes. Often we numerate the variable on the domain when it's too big to use different letters, for example for a domain of 5 variables we will use $x_1, x_2, ..., x_5$. A unique order of the variables is not consistent with all the possible networks in the domain, but if the context require precise order we will suppose that if x_n is a parent of x_m then $n < m$ in the order. We will denote the parent set of a variable x_n simply with Π_n instead of Π_{x_n} to lighten the notation.

In the context of discrete random variables we will denote with r_i the number of values the variable x_i can take. Similarly, we will denote the degrees of freedom of the parent set of a variable with

$$q_i = \prod_{x_l \in \Pi_i} r_l$$

With $\Pi_i = j$ we will mean that the parents of var. i assume the jth pattern of values on the possible total q_i.

Chapter 4

Structural Learning - Overview

4.1 Definition of the problem

We consider now the central problem of our study: the automatic learning of Bayesian Structure from observed data. We defined Bayesian Network and their components in previous chapter, but we did not formulate a fundamental question: how do we build a Bayesian Network? If we know an expert of the topic, we can ask him or her to incorporate his knowledge in a network. The problem is most of times this knowledge will not be enough or precise, and there are topics, like genetics, where we do not have any previous knowledge at all.

The alternative is try to build the network from the *observed data.* With data we usually mean a set of entries or samples, each one is in turn a set of values, one for each variable in the domain, taken from the same observation.

For example, if our domain is medical analysis, we will consider a certain number of random variables, one for each biological parameter we are considering (like cholesterol, number of white blood cells, etc.); every entry in our data set will contain a value for every parameter from a different patient. We make a simple example on how we consider data to be organized. Suppose we are considering the domain of the network exposed in 3.1, our data will be made of daily observation of the three simple events (rain, sprinkler and wet grass), and it can be represented in a table:

Sample N	Rain	Sprinkler	Wet Grass
1	T	F	T
2	F	F	F
3	F	T	T
4	F	T	F
5	T	T	T
6	F	F	F
...	...	...	...

In this case the variables are binary, but much more complex cases are possible. Sometimes there will be *missing values*: this is a vast topic itself and will not be covered here, so we will suppose to have complete data.

Building a network consist in two different problems: determinate the structure of network and the parameters, or the set Ξ of conditional mass functions. Usually the second is a minor problem and when the structure is defined it comes easily; many structure learning algorithms estimate parameters as part of their process. In general the parameters that are learned in a Bayesian network depend on the assumptions that are made about how the learning is to proceed.

We are now going to concentrate our attention on structural learning. Learning the structure of a Bayesian network can be considered a specific example of the general problem of selecting a probabilistic model that explains a given set of data and it has been proven to be NP-hard by Chickering [9]. Indeed, a simple look at the number of possible DAGs for a given number of nodes will indicate the problem is hard: the number of all possible DAGs with n nodes has been shown to be $O(n!2^{\binom{n}{2}})$ [10] !
We should remember that a priori we don't even know the number of edges, neither their orientation: the only constraint is that the structure must be an acyclic graph.

Despite the complexity results, various techniques have been developed to render the search tractable. All techniques however are usually regrouped in three main approaches [2]

- A constraint-based approach that uses conditional independences identified in the data

- A score and search approach through the space of Bayesian network structures
- A dynamic programming approach

As we will see for the previous works presented in the next chapter, the boundaries between groups are not always clear and many hybrid methods have been proposed. In the next sections we will briefly describe the characteristics of the three different main approaches. We will outline in detail the topics we will need in the next chapters.

4.2 Conditional Independence

A possible approach to learn the graph of a Bayesian Network is the use of statistical tests to recover the skeleton of the network structure and then orient the edges. A well known method is the PC algorithm by Spirtes and Glymour[11].

In general, to recover the skeleton of a network we need to identify the dependencies between the variables. In Bayesian Networks every variable/node is independent from its non descendants given its parents: it is not only necessary to consider two variables at a time, we will potentially need to consider also a set of possible neighbours for the variables.

Formally, given two variables x and y and a set of variable $\mathbf{z}$, we want to verify the hypothesis x and y are independent given the set $\mathbf{z}$, i.e $P(x, y|\mathbf{z}) = P(x|\mathbf{z})P(y|\mathbf{z})$. We indicate this assumption as $Ind(x, y|\mathbf{z})$, and the opposite assumption (i.e. dependence) as $Dep(x, y|\mathbf{z})$.

We can resume what said so far with the following theorem [19]:

Theorem 4.1 (Spirtes, Glymour and Scheines). *In a faithful BN* (G, X, Ξ)*there is an edge between the pair of nodes* $x, y \in X$ *iff* $Dep(x, y|\mathbf{z})$ *for all possible* $\mathbf{z}$.

4.2.1 Statistical Tests of Independence

Generally speaking, a statistical hypothesis test is a method of making decisions using data. In statistics, a result is called *statistically significant* if it is unlikely to have occurred by chance alone, according to a pre-determined threshold probability, *the significance level.* The phrase "test of significance" was coined by

Ronald Fisher: "Critical tests of this kind may be called tests of significance, and when such tests are available we may discover whether a second sample is or is not significantly different from the first" [36].

In frequency probability the test is done on a so called *null-hypothesis*, which typically corresponds to a general or default position. A statistical test answer then the following question: "Assuming that the null hypothesis is true, what is the probability of observing a value for the test statistic that is at least as extreme as the value that was actually observed?"[37]

In base of the probability returned from the test on a set of data , the null-hypothesis can be rejected or not (but never confirmed!). In our specific case, the null-hypothesis assumed is $Ind(x, y|\mathbf{z})$.

There are many possible independence tests, we present here the so called χ^2 and G-tests as in [13],[3], [38] and [18].

$$G^2 = 2\sum_{a,b,\mathbf{c}} N_{ab\mathbf{c}} \ln \frac{N_{ab\mathbf{c}}}{E_{ab\mathbf{c}}} \tag{4.1}$$

$$\chi^2 = \sum_{a,b,\mathbf{c}} \frac{(N_{ab\mathbf{c}} - E_{ab\mathbf{c}})^2}{E_{ab\mathbf{c}}} \tag{4.2}$$

Assuming independence, $E_{ab\mathbf{c}}$ is the expected number of samples where $x = a$, $y = b$ and $\mathbf{z} = \mathbf{c}$:

$$E_{ab\mathbf{c}} = \frac{N_{a\mathbf{c}} N_{b\mathbf{c}}}{N_{\mathbf{c}}} \tag{4.3}$$

where $N_{ab\mathbf{c}}$ is the number of times in the data where $x = a, y = b$ and $\mathbf{z} = \mathbf{c}$. In similar fashion we define $N_{a\mathbf{c}}$, $N_{b\mathbf{c}}$ and $N_{\mathbf{c}}$.

Both of this statistics are *asymptotically* distributed as the *chi-squared distribution* χ^2_{df} with degree of freedom

$$df = (D(x) - 1)(D(y) - 1) \prod_{w \in \mathbf{z}} D(w)$$

where $D(x)$ is the size of the domain of x. Using the χ^2 as a statistic test leads to the Pearson's χ^2 test of independence, while using the G leads to a likelihood ratio test of independence, also called a G-test[38].

The results of the test is called *p-value* and it is calculated as $1 - F(T_D)$, where F is the cumulative distribution function of χ^2_{df} and T_D is the value of the statistic for the data set D. The p-value corresponds to the probability of falsely rejecting the null hypothesis given that it is true. A problem of this kind of tests is that with a large conditioning set $\mathbf{z}$, any test $T(x, y|z)$ will return a high p-value with high probability leading to the acceptance (i.e. not rejecting) of $Ind(x, y|\mathbf{z})$.

Two heuristic solutions have appeared in literature. First, some algorithms using the above metrics [3] [13] [18] do not perform a test if they determine that there are not enough samples in the data set to achieve enough precision. The algorithms require that the average sample per count is at least some fixed number m. As in [13] we call this the *heuristic power rule.*

Second, several of the zero counts $N_{ab\mathbf{c}} = 0$ are heuristically declared as *structural zeros*, i.e. they are considered to be inherently part of the problem under consideration. Since they are supposed to be not free to vary, the degrees of freedom of the test should be adjusted. Spirtes *et al.* [18] subtract one from the the degree of freedom for each counter $N_{ab\mathbf{c}}$ equal to 0. Tsamardinos *et al.* [3] [13] instead consider a structural zero any case $N_{ab\mathbf{c}} = 0$ and also either the marginals are $N_{a\mathbf{c}} = 0$ or $N_{b\mathbf{c}} = 0$. Then if $N_{a\mathbf{c}} = 0$ they reduce then degree of freedom by $D(x) - 1$.

4.3 Search and Score

Search and Score algorithms consider the problem of learning Bayesian Networks as optimization problems. In order to find the *best* Bayesian Network two things are then needed; the first is a way to somehow quantify how "good" is a structure, which is usually done with a scoring function of some sort.

Definition 4.1 (Scoring Function)**.** *Formally we call scoring function, criterion or metric a function that takes in input a DAG and a data set and returns in output a value indicating how well the structure fits the data.*

The second element we need is obviously a procedure to efficiently explore the search space, which, we stress again, is exponential on the number of nodes. In the next subsections we will briefly recall the most used score functions, with particular attention to the BDeu metric, and make some general premise on the search methods.

4.3.1 Bayesian Dirichlet Score

One of the most used scoring function is the so called Bayesian Dirichlet metric, or BD metric for short. It was defined by Heckerman, Geiger and Chickering in 1995 [8]. As we have seen in chapter 2, and precisely with equation 2.9, what we can try to find an expression for the likelihood function and rank the structures using it: the greater the likelihood function, the greater the posterior probability, the better the structure for the given data.

The BD metric does exactly that: it gives a precise expression for likelihood function given a series of assumptions. With $x_i = k$ we will mean that variable x_i assume the kth value of his domain, which we consider ordered in some way. Similarly, given an order for all the possible combinations of values for the variables $t \in \Pi_i$, with $\Pi_i = j$ we indicate that the set Π_i presents the jth possible values combination. We are not going here to run through all the mathematical passages, we will limit ourselves to recall all the assumptions and the results. For the complete paper see [8].

Assumption 1 - Multinomlial Sample
Given a domain U and a database D, let D_l denote the first l-1 cases in the database. In addition, let x_{il} denote the variable x_i in the lth case and Π_{il} the corresponding parent set. Then, for all network structures B in U,

$$P(x_{il} = k | \Pi_{il} = j, D, G) = \theta_{ijk}$$

This first assumption says in practice that the probability for a variable to take a certain value depends only on the state of the parents.

Assumption 2 - Parameter Independence

- *The parameters associated with each variable in a network structure are independent*
- *The parameters associated with each state of the parents of a variable are independent*

Assumption 3. Parameter Modularity
Given two network structures G and G', if x_i has the same parents in G and G', then

$$P(x_i = k|\Pi_i = j, D, G) = P(x_i = k|\Pi_i = j, D, G')$$

In practice parameter modularity says that the probability for every variable to assume a certain value depends only on the local situation of the network, i.e. the parent set.

Example 4.1. In the following simple network, x_1 will have the same distribution, x_2 not:

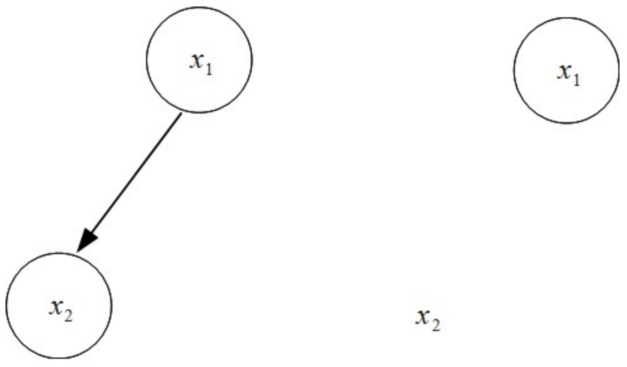


Figure 4.1

Assumption 4. Dirichlet
Given a network structure B and let be $\Theta_{ij} = \bigcup \theta_{ijk}$, $p(\Theta_{ij}|G)$ is Dirichlet for all Θ_{ij}. That is, there exists exponents N'_{ijk}, which depend on B, that satisfy

$$p(\Theta_{ij}|G) = c \prod_k \theta_{ijk}^{N'_{ijk}-1}$$

where c is a normalization constant.

This assumption give the name at the metric we are considering. The Dirichlet distribution has an important propriety: if the data have a multinomial distribution, and the prior distribution is distributed as a Dirichlet, then the posterior distribution of the parameter is also a Dirichlet. The exponents N'_{ijk} codify somehow the state of the information about the probabilities we are considering [8].

Assumption 5. Complete Data
The Database is complete. That is, it contains no missing data.

Given these assumptions, we can determinate a form for the likelihood function $P(D|G)$. Instead of the likelihood however we apply product rule once again and we observe that we can consider $P(D, G)$ equivalently. Thus, applying all the assumption we can write the **BD metric**:

$$P(D,G) = P(G) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij} + N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N'_{ijk} + N_{ijk})}{\Gamma(N'_{ijk})} \tag{4.4}$$

$\Gamma(x)$ is the Gamma function, which satisfies $\Gamma(1) = 1$ and $\Gamma(z+1) = z\Gamma(z)$, or in the integer case $\Gamma(z) = (z-1)!$.

N'_{ijk} are the parameters of the Dirichlet distribution and N_{ijk} is the number of samples in the observed data set such that x_i assumes the jth possible value of his domain and Π_i presents the kth possible combination of values of its elements.

We recall that r_i and q_i are the degrees of freedom for x_i and Π_i respectively, while $N_{ij} = \sum_k N_{ijk}$ and $N'_{ij} = \sum_k N'_{ijk}$; n is the number of variables in the domain.

The **BD score** is built taking the logarithm of equation 4.4. We write it explicitly

$$S_{BD}(D,G) = \log(P(G)) \sum_{i=1}^{n} \sum_{j=1}^{q_i} \left(\log \frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij} + N_{ij})} + \sum_{k=1}^{r_i} \log \frac{\Gamma(N'_{ijk} + N_{ijk})}{\Gamma(N'_{ijk})} \right) \tag{4.5}$$

Usually, the structure prior $P(G)$ is considered to be equal for all the possible graphs and omitted; in this case we speak of *uninformative prior*. Other approaches are possible, like penalizing somehow the number of edges: in [12] the structure prior used is 0.001^f where f is the number of free parameters in the DAG. The problem of the BD scoring function is to provide a value for all N'_{ijk}. A solution is to pose all $N'_{ijk} = 1$, which is known as the **K2** metric and was defined by Cooper and Herskovits in 1992 [39]. Alternatively, we can make further assumptions.

Assumption 6. Likelihood Equivalence
For any database D and two equivalent network structures G and G', the probability of D is the same.

Assumption 7. Structure Possibility
Given a domain U, $P(G) > 0$ *for all complete network structures G.*

Given the new assumptions, we can state [8]

$$N'_{ijk} = N'P(x_i = k, \Pi_i = j|G) \tag{4.6}$$

where N' is the only free hyperparameter and it is known as Equivalent Sample Size (ESS) and expresses the strength in our belief in the prior distribution. We call in this case the equations 4.4 and 4.5 the BDe metric and BDe score respectively, where "e" is for equivalence.

Unfortunately determining $P(x_i = k, \Pi_i = j|G)$ is still a huge problem. Thus is necessary to make a last assumption:

$$N'_{ijk} = N'P(x_i = k, \Pi_i = j|G) = \frac{N'}{r_i q_i} \tag{4.7}$$

Recalling that $N'_{ij} = \sum_k N'_{ijk}$, if the values of the ith variable in the domain assumes values $k = 1, 2, ..., r_i$ we can immediately also obtain:

$$N'_{ij} = \sum_k N'_{ijk} = \sum_k \frac{N'}{r_i q_i} = \frac{N'}{q_i} \tag{4.8}$$

The metric is called BDeu, where "u" is for uninformative. The BDeu score is the one that can be used in practice.

4.3.2 BIC and AIC Score

The Bayiesian Information Criterion, which was defined by Schwarz (1978) [7], and Akaike Information Criterion due to Akaike (1974) [6], are two other well known metrics based on likelihood function. They interpret a structure G as a set if independence constraints in the maximum-likelihood estimate, derived from the observed data, of a joint distribution over the domain.

The two score functions are very similar and differ only in the weight that is given to a penalty term based on the number of free parameters. We give here the equation from [5]

$$S_{BIC/AIC}(D,G) = \max_{\theta} L_{G,D}(\theta) - w\sum_{i=1}^{n}(q_i(r_i-1)) \tag{4.9}$$

For AIC $w = 1$, for BIC $w = \frac{\log N}{2}$, where N is the number of entries in the data set. $L_{G,D}$ is the log-likelihood function with respect to data D and structure G, and can be expressed as

$$L_{G,D}(\theta) = \log \prod_{i=1}^{n}\prod_{j=1}^{q_i}\prod_{k=1}^{r_i}\theta_{ijk}^{N_{ijk}} \tag{4.10}$$

Moreover, we know that

$$argmax_{\theta} L_{G,D}(\theta) = (\theta^*_{ijk})_{\forall ijk} = \left(\frac{N_{ijk}}{N_{ij}}\right)_{\forall ijk}$$

Using this information and moving the log inside the product the equation 4.10 can be re-written as follow:

$$S_{BIC/AIC}(D,G) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk}\log\frac{N_{ijk}}{N_{ij}} - w\sum_{i=1}^{n}(q_i(r_i-1)) \tag{4.11}$$

4.3.3 Decomposability and Score Equivalence

We introduce now two crucial proprieties for scoring metrics that are greatly useful during the search phase. The first is the so called *decomposability* of the score:

Definition 4.2 (Decomposable Score)**.** *Given a DAG G and a scoring function S(G), S is said to be decomposable if we can find a function s_i dependent only on the parent set of x_i in G such that*

$$S(G) = \sum_{i=1}^{n} s_i(\Pi_i)$$

It's immediate to check that all the BDeu, AIC and BIC metrics are decomposable. In fact observing the equations 4.8 and 4.11 we observe the scoring function contains a sum on all the variables in the domain, while all the parameters are dependent only on the parents of the ith variable in the sum, and the variable itself. The local score functions can be expressed as follows:

$$s_{BDeu}(x_i, \Pi_i, D) = \sum_{j=1}^{q_i} \left(\log \frac{\Gamma(\frac{N'}{q_i})}{\Gamma(\frac{N'}{q_i} + N_{ij})} + \sum_{k=1}^{r_i} \log \frac{\Gamma(\frac{N'}{r_i q_i} + N_{ijk})}{\Gamma(\frac{N'}{r_i q_i})} \right) \quad (4.12)$$

$$s_{BIC/AIC}(x_i, \Pi_i, D) = \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}} - w(q_i(r_i - 1)) \quad (4.13)$$

where in 4.12 we supposed uninformative prior.

We show the advantage of decomposability with a simple example.Suppose to have two DAG G and G' on the same set of variable, and suppose that the only difference between the two is a single edge $x_u \leftarrow x_v$ which be present in G' but absent in G.

Suppose also that we calculated $\forall i = 1, ..., n$ the local scores $s(x_i, \Pi_i, D)$ for the graph G; to calculate the score of G' we only need to locally score again only x_u:

$$S(G') = S(G) - s(x_u, \Pi_u, D) + s(x_u, \Pi'_u, D)$$

where $\Pi'_u = \Pi_u \cup \{x_v\}$. How useful is to have this property will be even more evident when we will explain the search procedures.

The second fundamental concept is *score equivalence.*

Definition 4.3 (Score Equivalence). *Given two equivalent DAG G and G' and a scoring function S, S is said to be score equivalent if and only if*

$$S(G) = S(G')$$

If we are searching for the best bayesian structure underling a certain distribution from our data, we obviously have not interest in discriminating DAGs which are equivalent, i.e. which describe the same conditional dependencies between the variables in the domain.

In other words, we are not normally interested in finding a specific DAG: every graph belonging to the same equivalence class can be a valid choice. Incidentally BIC, AIC and BDeu metrics are score equivalent. We omit a demonstration, but we recall that score equivalence is stated for BDeu in **Assumption 7**, i.e. is part if it's definition.

4.3.4 Search Algorithms: the space of DAGs and CPDAGs

During the years many procedures that make use of scoring function along with a search strategy of some sort have been proposed, but the general idea is to apply an heuristic to find structures with increasing score.

Most of the algorithms comprise, along with a scoring function, the following parts [2]

- a search space consisting of the various allowable states of the problem, each of which represents a Bayesian network structure
- a mechanism to encode each of the states
- a mechanism to move from state to state in the search space

Finding new structures means that we need to move in the space of all possible DAGs for a certain domain. To do it, we need certain operators. To cover the space of DAGs only three operators are needed [8]

- **Add Edge**
- **Remove Edge**
- **Invert Edge**

Operator *Add Edge* is used to insert an edge, with the only rule that the new graph must be still acyclic. *Remove Edge* on the other hand is always possible without contraindications. *Invert Edge* is used to invert an arc: in practice it first removes an edge $x \leftarrow y$ and then add the edge $x \rightarrow y$; again cycles must be checked. It is worth to point out that only Remove and Add Edge operators are necessary to explore all the possible DAGs, but a direct invert operation can be useful.

The use of a decomposable scoring function in this context reveals its full potential: at every step of an algorithm using the described operators it is necessary only the recalculation of the local score of the variables whose parent sets have changed.

The simplest algorithm that can be realised in a Search And Score context is a Greedy Local Search

Algorithm 6: Greedy Local Search

```
Data: data D, scoring function S
 G = empty graph
while The score in the last iteration has improved do
 |  apply the best change to G using Add, Remove and Invert Edge
end
return G
```

The first issue with such extremely simple approach is the high probability of getting stuck very fast at a local optimum. To avoid this problem,over the years more sophisticated algorithms have been proposed [2].

The presence of numerous networks with the same score, i.e. belonging to the same equivalent class, also pose a problem: many Invert Edge operations may be needed before we are able to escape the current equivalence class and finally find a new network with higher score using the Add or Remove Edge operators.

We explained in chapter 3 how equivalent classes of Bayesian Networks can be represented with PDAG, or more precisely how every class is represented by a unique complete PDAG, or CPDAG.

However, the operations to move in the space of CPDAG are more numerous and complicated; their complete definition, along with theorems for their efficient application are due to Chickering and can be found in [12]; we report here the list and the relative rules. We will say that an operator is valid if the resulting PDAG has a consistent extension.

In accordance with [12], we will use the following notation in the space of PDGAs: Π_x is the parent set of x, N_x is the set of neighbours of x, $N_{x,y}$ is the set of common neighbours of x and y and $\Omega_{x,y}$ is the set of parents of x that are neighbours of y. Moreover we denote by Z^{+x} the set $Z \cup \{x\}$ and with Z^{-x} the set $Z \backslash \{x\}$.

1. **InsertU.** Let x and y two nodes not adjacent in P. The insertion of the edge $x - y$ is valid if and only if every undirected path between x and y contains a node in $N_{x,y}$ and $\Pi_x = \Pi_y$. The score increase is:

$$s(y, N_{x,y} \cup \Pi_y^{+x}) - s(y, N_{x,y} \cup \Pi_y)$$

2. **DeleteU.** Let $x - y$ an undirected edge in P. The deletion of $x - y$ is valid if and only if $N_{x,y}$ is a clique of undirected edges. The score increase is:

$$s(y, N_{x,y} \cup \Pi_y) - s(y, N_{x,y} \cup \Pi_y^{+x})$$

3. **InsertD.** Let x and y two nodes not adjacent in P. The insertion of the edge $x \rightarrow y$ is valid if and only if every semi-directed path from y to x contains at least one node in $\Omega_{x,y}$, $\Omega_{x,y}$ is a clique of undirected edges and $\Pi_x \neq \Pi_y$. The score increase is:

$$s(y, \Omega_{x,y} \cup \Pi_y \cup x) s(y, \Omega_{x,y} \cup \Pi_y)$$

4. **DeleteD.** Let $x \rightarrow y$ an undirected edge in P. The deletion of $x \rightarrow y$ is valid if and only if N_y is a clique of undirected edges. The score increase is:

$$s(y, N_y \cup \{\Pi_y \backslash x\}) - s(y, N_y \cup \Pi_y)$$

5. **InvertD.** Let $x \rightarrow y$ an undirected edge in P. The reversal of $x \rightarrow y$ is valid if and only if every semi-directed path from x to y that does not include the edge $x \rightarrow y$ contains at least one node in $\Omega_{x,y} \cup N_{x,y}$ and $\Omega_{x,y}$ is a clique of undirected edges. The score increase is:

$$s(y, \Pi_y^{-x}) + s(x, \Pi_x^{+y} \cup \Omega_{x,y}) - s(y, \Pi_y) - s(x, \Pi_x \cup \Omega_{x,y})$$

6. **MakeV.** Let $x - y - z$ be any length-two undirected path in P such that x and y are not adjacent. Replacing the undirected edges with directed edges to create the v-structure $x \rightarrow y \leftarrow z$ is valid if and only if every undirected path between x and y contains a node in $N_{x,y}$. The score increase is:

$$s(z, \Pi_z^{+x,y} \cup \{N_{x,y}^{-z}\}) + s(z, \Pi_z \cup \{N_{x,y}^{-z}\}) - s(z, \Pi_z^{+x} \cup \{N_{x,y}^{-z}\}) - s(z, \Pi_z \cup N_{x,y})$$

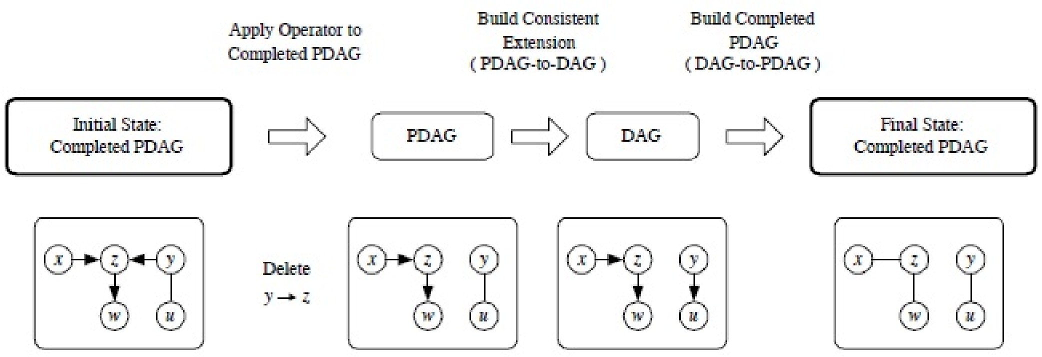

Apply Operator to Completed PDAG
Build Consistent Extension (PDAG-to-DAG)
Build Completed PDAG (DAG-to-PDAG)
Initial State: Completed PDAG
PDAG
DAG
Final State: Completed PDAG
Delete
y → z
x
z
y
w
u

Figure 4.2: Schema showing the procedure explained in [12] to explore the space of CPDAGs

After applying an operator to a CPDAG, the resulting PDAG is not necessary complete. To recover the complete PDAG representing the equivalence class resulting from the operator, it's necessary to use the transformation algorithms presented in chapter 3.

The schema of the entire operation, as presented in [12] is the presented in figure 4.2. The main advantage of using the PDAG is that we need to search in a much smaller space, at the price of more complex operations.

4.3.5 Search Algorithms: examples

To conclude, we give a fast overview of algorithms used to search both in the space of DAGs and CPDAGs. A complete study can be found in [2].

Greedy search has been abundantly used; some of the earliest work that looked at greedy methods to learn Bayesian network structure was by Herskovits and Cooper in 1991 [39] and 1992 [40].

The authors provided a way to construct a Bayesian network structure given a data sample and an ordering of the various variables and used a Bayesian scoring criterion, which has come to be known as the K2 score. More greedy algorithms using an order for the variables have followed [41] [42] [43]. Other approaches still use greedy procedures, but without ordering the variables [8] [44] [45].

Between the others algorithm used, in literature we can find evolutionary algorithms, as in [46], simulated annealing, as in [47], or particle swarm optimization, as in [48]. Others successfully used the CPDAG search space, a recent example is Ant Colony Optimization due to Daly and Shen [49].

4.4 Dynamic Programming

Aside from the two major techniques of structure learning that have been discussed, there is a third method that is similar to the score-and-search approach, but does not have the search aspect [2]. These methods use dynamic programming to compute optimal models for a small set of variables and in some cases combine these models.

A well-known algorithm is the one proposed by Silander and Myllymaki in 2006 [50]. Other similar approaches have been proposed by Koivisto and Sood (2004) [51], Eaton and Murphy [52], Perrier *et al.* (2008) [14].

We briefly recall here the Silander-Myllymaki algorithm. We call *sink* of a network a node without outgoing edges: given the acyclicity a sink must exists; then the algorithm follows the following schema:

1. Calculate the local scores for all $n2^{n-1}$ different (variable, variable set)-pairs.
2. Using the local scores, find best parents for all $n2^{n-1}$ (variable, parent candidate set)-pairs.
3. Find the best sink for all 2^n variable sets.
4. Using the results from Step 3, find a best ordering of the variables.
5. Find a best network using results computed in Steps 2 and 4.

The problem with this kind of algorithms should be evident without any further analysis. Such procedures require big amount of resources, time and specially memory (the original Silander Mylimaki implementation the memory requirement was 2^{n+2} bytes), and become infeasible very fast with the increasing number of variable (the limit is normally around 30). On the other hand, they are complete algorithms and provide the global optimum solution for the problem, which is not guaranteed by heuristic procedures [2].

Chapter 5

Previous Works

5.1 The Max-Min-Hill-Climbing

In 2006 Tsamardinos, Brown and Aliferis presented in [3] an hybrid algorithm for automatic Bayesian Network structure learning.

The method can be divided in two phases. The first phase, which is called the *Max-Min-Parent-Children* algorithm (MMPC for short), use statistical test of conditional independence to find the probable sets of parents and children for every variable x in the domain, similarly to the Sparse Candidate algorithm [53].

The second phase is a simple greedy local search applied in the space of the DAGs but restricted to the hypothetical skeleton identified by phase one. Thus we can say that the Max-Min-Hill-Climbing (MMHC) algorithm is an hybrid between the conditional constraints and search & score approaches. In the next sections we will outline all the part of the algorithms involved.

5.1.1 Max-Min-Parent-Children Algorithm

The Bayesian network learning algorithm presented in [3] is based on the local discovery algorithm called Max-Min Parents and Children used to reconstruct the skeleton of the Bayesian network. The *Max-Min-Parent-Children* Algorithm (MMPC) algorithm try to find for each variable x in the domain a set, denoted as *CPC*, containing all the nodes candidates to be parents or children of x. It is worth recalling that two equivalent structure, i.e. representing the same distribution, will have the same skeleton, thus the same sets of parents-children for each variable in the domain.

The Max-Min part of the algorithm name refers, as we will see, to the specific heuristic used. MMPC runs on a target variable t and provides a way to identify the existence of edges to and from t, without identifying the orientation of the edges.

By invoking MMPC with each variable as the target one can identify all the edges (in an un-oriented fashion) in the network, i.e. identify the skeleton of the Bayesian network, using statistical tests of independence (see 4.2.1).

We report here the algorithms 7 and 8 as presented in the original paper, we add algorithm 9 to simplify the explanation:

Algorithm 7: MMPC

```
Data: target variable $t$, data D
Result: CPC for $t$
% Phase 1: forward
CPC = empty set
while It's possible to add a variable in CPC do
    $(f, assoc_f) = \text{MaxMinHeuristic}(t, CPC)$
    if $assoc_f \neq 0$ then
        add f to CPC
    end
end
% Phase 2: backward
for all $x \in CPC$ do
    if $\exists \mathbf{s} \subset CPC$ s.t. $Ind(x, t|\mathbf{s})$ then
        remove x from CPC
    end
end
return CPC
```

```
Algorithm 8: MaxMinHeuristic
Data: target variable $t$, sub set of variables CPC
Result: the maximum over all the variables of the minimum association
        with $t$ relative to CPC, and the variable that achieve the
        maximum
$assoc_f = max_x MinAssoc(x, t|CPC)$
$f = argmax_x MinAssoc(x, t|CPC)$
return $f, assoc_f$
```

```
Algorithm 9: MinAssoc
Data: variables $t, x$, sub set of variables CPC
Result: the minimum association between $t, x$ given CPC
$assoc = \infty$
for all possible subsets $\mathbf{s}$ of CPC do
    get the p-value from the statistical test $T(t, x|\mathbf{s})$
    if p-value$> \alpha$ then
        return 0
    end
    if $-\log($ p-value$) < assoc$ then
        $assoc = -\log($ p-value$)$
    end
end
return $assoc$
```

Algorithm 7 works in this way: the CPC for a variable t is initially created empty; then, at every step, a couple formed by a variable f and a value representing its *association* with t given the current CPC is returned by the *Max-Min-Heuristic.* If the association for f is not zero, then f is added to the CPC and the process re-iterated. If not, the algorithm starts a backward phase: all variables in the CPC are tested against all possible subsets $\mathbf{s}$ of CPC: if there exists a $\mathbf{s}$ such that $Ind(x, t|\mathbf{s})$, then the variable is eliminated from the CPC.

The algorithm *per se* is simple, the key is to understand how the Max-Min-Heuristic works exactly. Given a variable t and a set of variable CPC, the Max-Min-Heuristic select the variable x which maximize the *association* with t

between all the variables in the domain not already in the CPC.

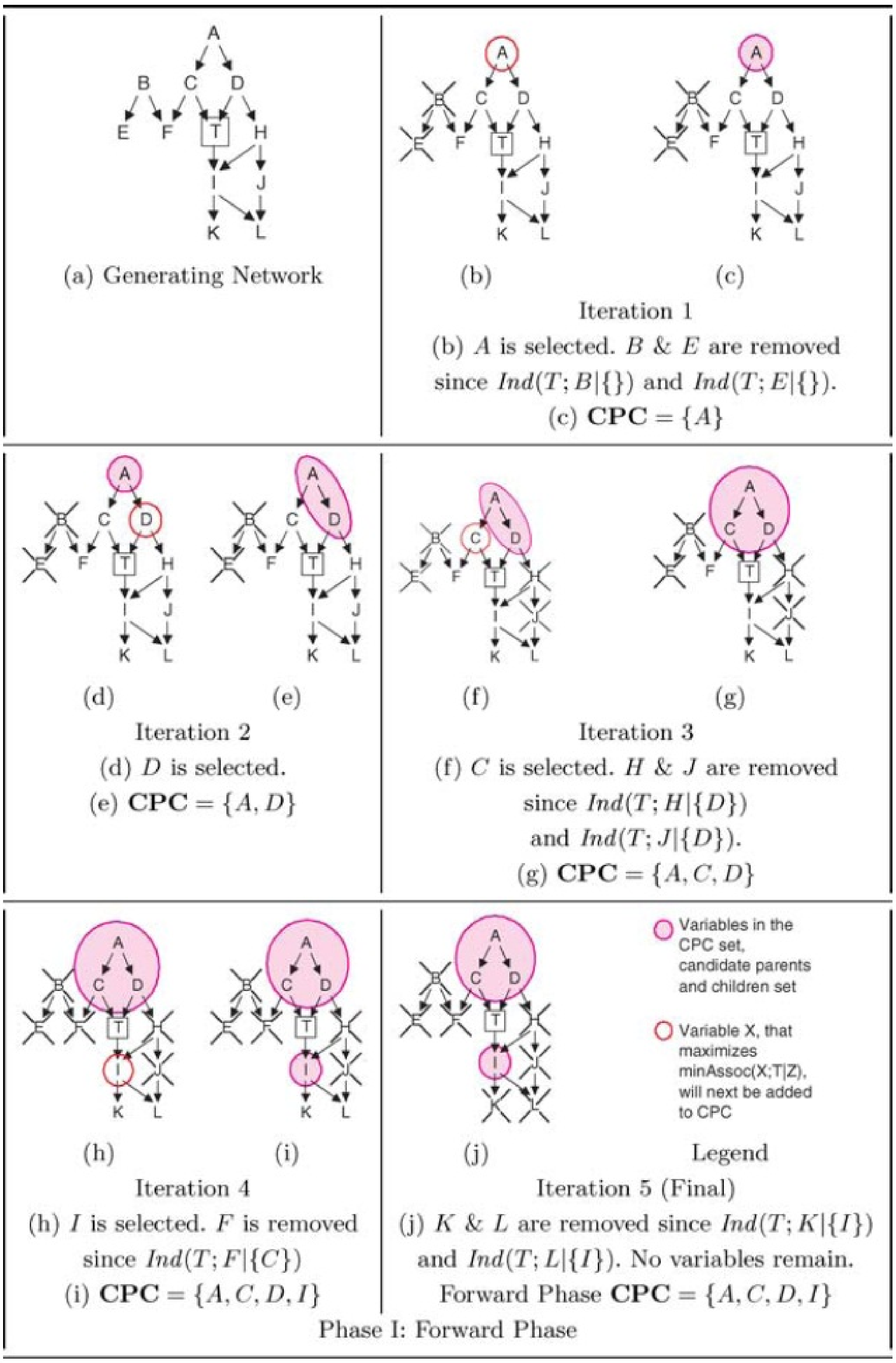
The *association* considered for each variable is the minimum association it is possible to find: for all possible subsets **s** of CPC, the procedure (algorithm 9) $MinAssoc(x, t|CPC)$ executes a statistical test of independence, in the original paper the G-test we presented in section 4.2.1.

If the p-value which corresponds to the probability of falsely reject the hypothesis $Ind(x, t|S)$ is less than the significance level α, the variable x and t are considered independent given **s** and the association between x and t is 0, else the association is calculated as $-\log($ p-value$)$.

Recapping, at every step of the forward phase the MMPC algorithm add to the CPC the variable which maximize the minimum association with t, which is found performing independence tests over all the possible subsets of the current CPC. The max-min heuristic is admissible in the sense that all variables with an edge to or from T and possibly more will eventually enter CPC [13]. The intuitive justification given for the heuristic is to select the variable that remains highly associated with t despite the best efforts (i.e., after conditioning on all subsets of CPC) to make the variable independent of t. Phase 1 stops when all remaining variables are independent of the target t given some subset of CPC, i.e., the maximum minimum association reaches zero.

The backward phase is necessary because we have not any grantee that variables are inserted in the CPC in a correct order: it may happen that variable x would be considered independent by the algorithm given variable y, but is added to the CPC before y, thus we need to ensure that at least $Ind(x, t|y)$ is tested at a later time.

Example 5.1. We report from the original paper [13] an example trace of the algorithm with target node T. For the example it is assumed that the sample size is large enough for the results of all tests of conditional independence performed by MMPC to return the correct result. Thus, the results of each call to $Ind(X; T|Z)$ can be assessed simply by looking at the graph. In figure 5.1 it is showed the forward phase, in figure 5.2 the backward phase.


A
B
C
D
E
F
T
H
I
J
K
L
Variables in the CPC set, candidate parents and children set
Variable X, that maximizes minAssoc(X;T|Z), will next be added to CPC
Legend

(a) Generating Network

(b) (c)

Iteration 1

(b) A is selected. B & E are removed since $Ind(T; B|\{\})$ and $Ind(T; E|\{\})$.

(c) **CPC** $= \{A\}$

(d) (e)

Iteration 2

(d) D is selected.

(e) **CPC** $= \{A, D\}$

(f) (g)

Iteration 3

(f) C is selected. H & J are removed since $Ind(T; H|\{D\})$ and $Ind(T; J|\{D\})$.

(g) **CPC** $= \{A, C, D\}$

(h) (i)

Iteration 4

(h) I is selected. F is removed since $Ind(T; F|\{C\})$

(i) **CPC** $= \{A, C, D, I\}$

(j)

Iteration 5 (Final)

(j) K & L are removed since $Ind(T; K|\{I\})$ and $Ind(T; L|\{I\})$. No variables remain.

Forward Phase **CPC** $= \{A, C, D, I\}$

Phase I: Forward Phase

Figure 5.1: Example trace of MMPC from [3]: forward phase

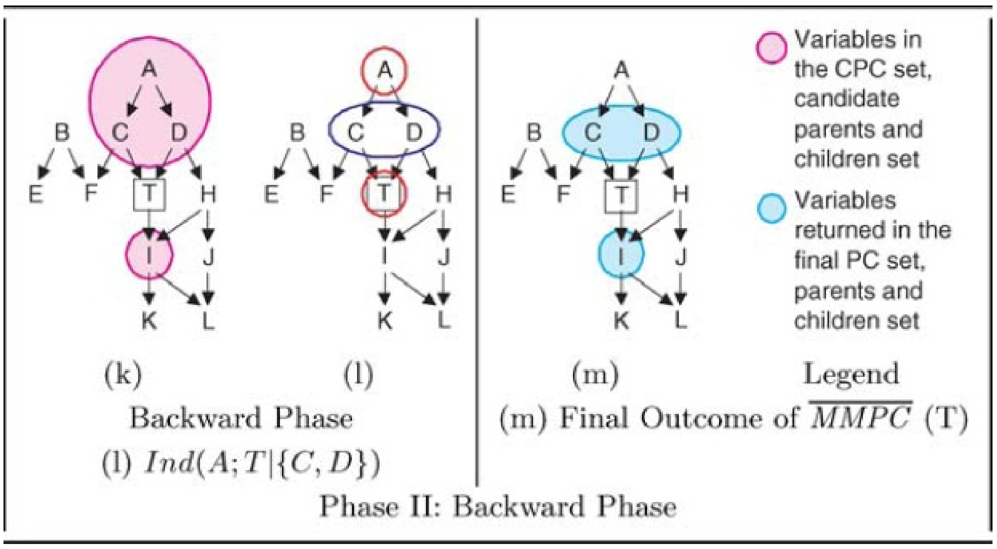

A
B
C
D
E
F
T
H
I
J
K
L
(k)
(l)
(m)
Legend
Variables in the CPC set, candidate parents and children set
Variables returned in the final PC set, parents and children set
Backward Phase
(l) $Ind(A;T|\{C,D\})$
(m) Final Outcome of $\overline{MMPC}$ (T)
Phase II: Backward Phase

Figure 5.2: Example trace of MMPC from [3]: backward phase

5.1.2 Avoiding False Positive in MMPC

In an ideal scenario, MMPC will return no false negatives, i.e., will include all the parents and children of t in its output. The MMPC algorithm however has a problem: it may include false positive, i.e include in the CPC nodes that are not either parent or children of the target variable, even in the ideal setting.

We do not give here the exact explanation, which is given in the original paper and requires additional theorems and definitions; generally speaking the problem is due to some particular configurations of edges in the graph.

The solution, however, is quite straightforward: if in the network underlying the data a variable x is the parent of y, the algorithm will (hopefully) insert x in CPC of y; obviously, being y one of the children of x, the algorithm will also include y in the CPC of x. In other words, all the CPC must be symmetric, but false positive are not [13].

We call the final algorithm, which will find the CPC for all the variables in the domain, MMPC'. We present here the pseudo-code.

```
Algorithm 10: MMPC'
Data: data set D
Result: the CPCs for each variable in the domain, without false positive
for each variable $x$ in the domain do
    $CPC_x$ = MMPC($x$,D)
end
for every ordered couple of variables $(x, y)$ in the domain do
    if $x \in CPC_y$ and $y \notin CPC_x$ then
        remove $x$ from $CPC_y$
    end
end
return all the CPCs found
```

5.1.3 Permutation Independence Test

As we explained, the G^2 and χ^2 statistics we introduced in 4.2.1 are asymptotically distributed as χ^2_{df} with a certain number of degree of freedo, i.e. the returned p-value is approximate and converges to the true value in the sample limit.

"Statisticians have long warned that the approximation is often poor in many circumstances, particularly when the sample size is low or the probabilities of the distribution are extreme. Ideally, one would prefer to use exact tests of independence. Unfortunately, in the general case such tests have a high computational overhead that prohibits their use in the context of learning large graphical models. In addition, they require highly specialized software that is often proprietary" [13].

Let consider again two variable, x and y, of which we want to test the independence over the marginal set $\mathbf{z}$. As we did before, we call $N_{ab\mathbf{c}}$ the number of times in the data set where $x = a, y = b$ and $z = c$. $N_{b\mathbf{c}}$, $N_{a\mathbf{c}}$ and $N_{\mathbf{c}}$ are defined in similar fashion.

"A mainstream approach to exact testing is called the *exact conditional approach* that considers the row and column marginals in each table $N_{a\mathbf{c}}$, $N_{b\mathbf{c}}$, and $N_{\mathbf{c}}$ as fixed. The distribution of the test statistic under the null hypothesis is then calculated conditioned on these marginals. Specifically, to calculate the exact p-value one needs to calculate $P(T_0 \geq T | Ind(x, y|\mathbf{z}))$ where T_0 is the observed test statistic. This in turn implies identifying all contingency tables with the same marginals and whose test statistic is larger or equal to the observed one. The number of possible tables with the same marginals quickly explodes" [13]

Recently, to improve the precision of the test for low sample size data sets Tsamardinos *et al.* proposed instead [13] an adjustable permutation method.

"Notice that, each table (where $\mathbf{z} = \mathbf{c}$) with the same marginals as the observed table can be produced by permuting the values of x or y of the samples (while retaining $\mathbf{z} = \mathbf{c}$). For example, for binary variables x, y, z suppose we have the observations $(0, 1, 0)$ and $(1, 0, 0)$ giving $N_{x=0,z=0} = N_{x=1,z=0} = N_{y=0,z=0} = N_{y=1,z=0} = 1$.

Permuting the two values of y between the only two observations provides the permuted data $(0, 0, 0)$ and $(1, 1, 0)$ with the same marginals. Under the null hypothesis of independence, this is justified as follows: since x and y are assumed independent given z, any such permutation has the same probability of being observed.

Calculating all such possible permutations is equivalent to enumerating all possible tables with the same marginals. However, one can sample from the space of all possible permutations (tables) randomly to estimate $\overline{P}(T_0 \geq T|Ind(x, y|z))$. Such methods are called Monte Carlo Permutation methods [54]. " [13]

We denote with D_0 the observed data. We obtain permuted data $D_i, i > 0$ as follows: for each possible value $\mathbf{c}$ of $\mathbf{z}$, randomly permute the values of y in D_0 only among the cases where $\mathbf{z} = \mathbf{c}$ (i.e., ensuring all marginals remain the same). We denote with $T(Di)$ the value of the statistic (either χ^2 or the G statistic) on the data D_i. The basic procedure is shown in the following Algorithm:

```
**Algorithm 11:** Basic Permutation
**Data**: data set $D_0$, variables $x$,$y$, set of marginals $\mathbf{z}$, number of
        permutation B
**Result**: p-value
**for** $i = 1, .., B$ **do**
    Randomly permute $D_0$ to obtain $D_i$
    Calculate $T(D_i)$
**end**
**return** $\#\{T(D_0) \leq T(D_i), i = 1, ..., B\}/B$
```

Tsamardinos *et al.* reported a sufficient number of permutations B to be in a range between 1000 to 5000, which makes the procedure quite costly for learning large graphical models. To improve the computational requirements, the authors designed an adjustable procedure that may stop early the computation of more permuted statistics. The procedure infers whether the current approximation of the p-value is sufficiently close to the true p-value to make a decision at significance level α.

The first rule implemented in the algorithm checks whenever a simple test on the not-permuted data is enough to make a decision; if a conservative asymptotic test (specifically the G test) returns a relatively low p-value (lower than 0.001 in Tsamardinos experiments) dependence is immediately accepted. Similarly, if a liberal asymptotic test (the χ^2 statistic) returns a high p-value (larger than 0.5), independence is immediately accepted instead:
Rule 1 if $p_g < 0.001$ return Dep., else if $p_{\chi^2} > 0.5 returnInd.$

The more complicated algorithms, conditions and rules will not be recalled here, and can be found in the original paper [13].

5.1.4 Hill Climbing

To conclude our fast excursus on the MMHC algorithm we present the simple greedy local search adopted in the original paper [3]. It is a simple Hill Climbing in the space of DAGs, starting from an empty graph and using the BDeu scoring function as presented in equation 4.5. At every step one of the possible operators is used (*Add Edge, Remove Edge and Invert Edge*). Obviously the resulting graph must remain acyclic at every step, and *Add Edge* is used only if the interested variables belong to their reciprocal CPCs. At every step, the operation on the graph that bring to the highest possible score is used; the score can decrease, but if 15 iterations passes without a global improvement the algorithm stops and the best graph found is returned. A TABU list contains the last 100 graphs visited and it is used to avoid returning on already considered solutions.

```
Algorithm 12: Max Min Hill Climbing
Data: data set D
Result: p-value
Perform MMPC on D
G = empty graph
L = empty list of graphs
while less then 15 iterations has passed from the last improvement in score
do
    Apply at G the operator leading to the best possible DAG G' such that:
    - G' is not in L
    - G' doesn't have any edge contradicting the CPCs
    Put G' in L
    Remove from L the oldest graph if $size(L) > 100$
end
return the best graph found
```

5.2 Complete Branch-and-Bound Algorithm

As we briefly explained in 4.4, the dynamic programming based methods to find the optimal structure are strongly limited in the number n of variables in the domain: they spend time and memory proportional to $n2^n$. Such complexity forbids the use of those methods to a couple of tens of variables, mainly because of the memory usage, while time complexity is also a issue.

Many methods have been proposed trying to improve the dynamic programming method to work over reduced search spaces. On a different front, Jaakkola *et al.* [55] applied a linear programming relaxation to solve the problem, together with a branch-and-bound search. We consider an algorithm that also uses branch and bound proposed by de Campos and Ji, which employs a different technique to find bounds and showed that branch and bound methods can handle somewhat larger networks than the dynamic programming ideas [5].

The method can be divided in two phases. First a cache containing the scores (AIC, BIC or BDeu) for all variables and possible parent sets are calculated; a series of mathematical rules are used to avoid computations and reduce the search space. Then a B&B procedure is applied. In the next sections we resume both phases.

5.2.1 Cache Construction and Score Properties

"Local scores need to be computed many times to evaluate the candidate graphs when we look for the best graph. Because of decomposability, we can avoid to compute such functions several times by creating a cache that contains $s_i(\Pi_i)$ for each x_i and each parent set Π_i. Note that this cache may have an exponential size on n, as there are 2^{n-1} subsets of $\{x_1, ..., x_{i-1}, x_{i+1}, ..., x_n\}$ to be considered as parent sets. This gives a total space and time of $O(n2^n\nu)$ to build the cache, where ν is the worst-case asymptotic time to compute the local score function at each node"[5].

Instead, de Campos and Ji described a collection of mathematical results for AIC, BIC and BDeu that can be used to obtain much smaller caches in many practical cases[5]. We do not report here the proves of the lemmas; the reader should refer to the complete discussion in [5].

The first fundamental simple lemma, stated among others in [56], holds for all three the scoring function described in 4.3 and it is the following:

Lemma 5.1. *Let x_i be a node of G', a candidate DAG for a Bayesian network where the parent set of x_i is Π'_i. Suppose $\Pi_i \subset \Pi'_i$ is such that $s_i(\Pi_i) > s_i(\Pi'_i)$ (where s is one of BIC, AIC, BD or derived criteria). Then Π'_i is not the parent set of x_i in an optimal DAG G^*.*

"Unfortunately lemma 5.1 does not tell us anything about supersets of Π'_i, that is, we still need to compute scores for all the possible parent sets and later verify which of them can be removed. This would still leave us with $n2^n\nu$ asymptotic time and space requirements (although the space would be reduced after applying the lemma)" [5].

There are a series of more strict proprieties, two separated series for BIC/AIC and BDeu. We report here the statements of BIC/AIC theorems:

Theorem 5.1. *Using BIC or AIC as score function, suppose that x_i, Π_i are such that $q_i > \frac{N}{w}\frac{\log r_i}{r_i-1}$. If Π'_i is a proper superset of Π_i , then Π'_i i is not the parent set of x_i in an optimal structure.*

Corollary 5.1. *Using BIC or AIC as criterion, the optimal graph G has at most $O(\log N)$ parents per node.*

Theorem 5.2. *Let BIC or AIC be the score criterion and let x_i be a node with $\Pi_i \subset \Pi'_i$ two possible parent sets such that $q'_i(r_i - 1) + s_i(\Pi_i) > 0$. Then Π'_i and all supersets $\Pi''_i \supset \Pi'_i$ are not optimal parent configurations for x_i.*

Theorem 5.1 and Corollary 5.1 ensures that the cache stores at most $O(\sum_{t=0}^{\lceil \log N \rceil} \binom{n-1}{t})$ elements for each variable (all combinations up to $\lceil \log N \rceil$ parents) [5]. Theorem 5.2 is quite useful in practice because it is applicable even in cases where Theorem 5.1 is not, implying that fewer parent sets need to be inspected [5] and it provides a bound to discard parent sets without even inspecting them.

BD theorems are more numerous and requires additional notations. However only the last one, which we recall here, is used in practice:

Theorem 5.3. *Let $K_{ij}^{\Pi_i} = \{1 \leq k \leq r_i : N_{ijk}\}$ be the indices of the categories of x_i such that $N_{ijk} \neq 0$ and $K_i^{\Pi_i} = \cup K_{ij}^{\Pi_i}$. Given the BDeu score and two parent sets Π_i^0 and Π_i for a node x_i such that $\Pi_i^0 \subset \Pi_i$ and $N'_{ij} =\leq 0.8349$ for every j* *for the parent set* Π_i*, if* $s_i(\Pi_i^0) > -|K_i^{\Pi_i}| \log r_i$ *then neither* Π_i *nor any superset* $\Pi_i' \supset \Pi_i$ *are optimal parent sets for* x_i*.*

"Theorem 5.3 provides a bound to discard parent sets without even inspecting them because ... the idea is to check the validity of Theorem 5.3 every time the score of a parent set Π_i of x_i is about to be computed by taking the best score of any subset and testing it against the theorem"[5].

Whenever possible during the cache construction, the algorithms discards Π and do not even look into all its supersets.

De Campos notes that the assertion $N'_{ij} \leq 0.8349$ required by the theorem is not too restrictive, because as parent sets grow, ESS is divided by larger numbers. Hence, the values N'_{ij} become quickly below such a threshold. It's worth noting that the theorems used in practice depends also on the number of samples in the data set.

5.2.2 Search Algorithm

After the cache has been built, the reduced search space is inspected using a *branch-and-bound algorithm* to find the best structure. The algorithm uses a relaxation of the problem: also graphs with cycles are considered during the search.

The method accepts in inputs also constraints on the structure of the network; specific edges can be set as mandatory or forbidden, or there may be limit to the number of parents a specific variable can have, etc.

We give here the algorithm from the original paper [5] The initialization of the algorithm is as follows:

- $C : (x_i, \Pi_i) \to R$ is the cache with the scores for all the variables and their possible parent configurations. This is constructed using a queue and analyzing parent sets according to the properties previously introduced, which saves (in practice) a large amount of space and time. All the structural constraints are considered in this construction so that only valid parent sets are stored.

- G is the graph created by taking the best parent configuration for each node without checking for acyclicity (so it is not necessarily a DAG), and s is the score of G. This graph is used as an upper bound for the best possible graph, as it is clearly obtained from a relaxation of the problem (the relaxation comes from allowing cycles).

- H is an initially empty matrix containing, for each possible arc between nodes, a mark stating that the arc must be present, or is prohibited, or is free (may be present or not). This matrix controls the search of the B&B procedure. Each branch of the search has a H that specifies the graphs that still must be searched within that branch.

- Q is a priority queue of triples (G, H, s), ordered by s (initially it contains a single triple with G, H and s as mentioned. The order is such that the top of the queue contains always the triple of greatest s, while the bottom has the triple of smallest s.

- (G_{best}, s_{best}) keeps at any moment the best DAG and score found so far. The value of s_{best} could be set to $-\infty$, but this best solution can also be initialized using any inner approximation method. For instance, the algorithm use a procedure that guesses an ordering for the variable, then computes the global best solution for that ordering, and finally runs a hill climbing over the resulting structure. All these procedures are very fast (given the small size of the precomputed cache that we obtain in the previous steps). A good initial solution may significantly reduce the search of the B&B procedure, because it may give a lower bound closer to the upper bound defined by the relaxation (G, H, s).

- *iter*, initialized with zero keeps track of the iteration number. *bottom* is a user parameter that controls how frequent elements will be picked from the bottom of the queue instead of the usual removal from the top. For example, a value of 1 means to pick always from the bottom, a value of 2 alternates elements from the top and the bottom evenly, and a large value makes the algorithm picks always from the top.

The main loop of the B&B search is as follows:

- While Q is not empty, do

  1. Increment $iter$. If $\frac{iter}{bottom}$ is not an integer, then remove the top of Q and put into $(G_{cur}, H_{cur}, s_{cur})$. Otherwise remove the bottom of Q

into $(G_{cur}, H_{cur}, s_{cur})$. If $s_{cur} \leq s_{best}$ (worse than an already known solution), then discard the current element and start the loop again.

2. If G_{cur} is a DAG, then update (G_{best}, s_{best}) with (G_{cur}, s_{cur}), discard the current element and start the loop again (if G_{cur} came from the top of Q, then the algorithm stops: no other graph in the queue can be better than G_{cur}).
3. Take a cycle of G_{cur} (one must exist, otherwise we would have not reached this step), namely $(x_{a_1} \rightarrow x_{a_2} \rightarrow ... \rightarrow x_{a_{q+1}})$, with $a_1 = a_{q+1}$.
4. For $y = 1, ..., q$, do
   (a) Mark on H_{cur} that the arc $x_{a_y} \rightarrow x_{a_{y+1}}$ is prohibited. This implies that the branch we are going to create will not have this cycle again.
   (b) Recompute (G, s) from (G_{cur}, s_{cur}) such that the new parent set of $x_{a_{y+1}}$ in G complies with this new H_{cur}. This is done by searching in the cache $C(x_{a_{y+1}}, \Pi_{a_{y+1}})$ for the best parent set. If there is a parent set in the cache that satisfies H_{cur}, then include the triple (G, H_{cur}, s) into Q.
   (c) Mark on H_{cur} that the arc $x_{a_y} \rightarrow x_{a_{y+1}}$ must be present and that the sibling arc $x_{a_y} \leftarrow x_{a_{y+1}}$ is prohibited, and continue the loop of step 4. *(Step 4c forces the branches that we create to be disjoint among each other.)*

The algorithm is correct and complete, i.e. it explores all the search space [5]. The algorithm was able to outperform the state-of-the-art dynamic programming, and it is also an any time procedure: if stopped before the optimum is found, it will be able to return an upper bound, given by the score of the graph at the first position of Q, and a lower bound, given by the best DAG found so far. The stopping criterion might be based on number of steps, time consumption and/or percentage of error.

Chapter 6

Hybrid Method using Statistical Constraints

In this chapter we expose the main idea and contribution of this thesis. In the next sections we describe the motivations behind our work and we detail the procedure proposed, then we give some indications on implementation and, finally, we outline the experiment settings, we discuss the results and we compare them with related works.

6.1 Hybrid Algorithms

As explained in chapter 5, the B&B algorithm by de Campos and Ji use constraints on the edges, created during iterations, to efficiently explore the search space.

However, it is possible to impose structural constraints in input, in the form of a matrix of nxn dimension, where every single edge $x_j \rightarrow x_y$ is represented by position (i, j) and can be set as forbidden (using 1), mandatory (using -1) or free (using 0). It is also possible to give more complex constraints as limiting the number of parent for each variable.

The authors tested the algorithm with randomly generated networks and used random structural constraints valid for the networks. They verified substantial gain in term of cache dimension and execution time and the algorithm was able to find the optimal solution with networks up to 70 nodes, and approximate solutions for 100 nodes [5].

The possibility to give constraints in input may be useful when the problem is restricted to search for a specific structure, or to incorporate experts knowledge of the domain. For example, we may know that some variables represent external factors and thus the corresponding nodes should not have incoming edges.

Obviously, in many cases our knowledge of the domain is very limited and it will be not enough to infer directly a large number of structural constraints, neither we can impose them at random. The MMPC algorithm, described in chapter 5, however, *de facto* imposes constraints that we can treat as *valid with a certain probability.*

Thus, we decided to exploit the options given by the B&B algorithm and add to the complete search the knowledge derived by the MMPC algorithm. In other words, we first calculate for a certain network and data the set of possible parents or children for each node, then we use the same data as input for B&B program, forbidding all edges $x \leftarrow y$ such that $x \notin CPC(y)$, i.e. building the constraints input matrix with a 1 at every position (i, j) such that $x_i \notin CPC(x_j)$, and 0 for all the other positions.

The main problem of statistical tests, particularly with small sample size data sets, is that they can introduce false negatives: edges that are part of the best network may be excluded by the MMPC algorithm and thus they will not be present in the structure obtained using CPCs as constraints.

In order to try to recover eventually missing edges we decide to add a post-processing phase: a unconstrained fast search algorithm on the structure found by the B&B algorithm. The final procedure we wanted to adopt is summarized in figure 6.1.

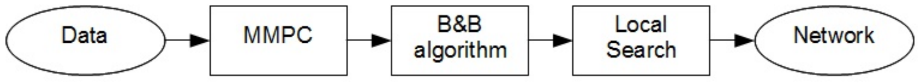


Figure 6.1: Procedure Diagram

We wanted to test the hybridization of the MMPC algorithm with the complete state-of-the-art B&B algorithm and with a greedy search at the top in order of precisely evaluate the drawbacks in term of quality, along with the benefits in term of time and memory; we expected the latter to be sensible, however we did not know what to expect from score. Our hypothesis was that the if constraints were good enough to allow the B&B algorithm to reach a network near enough to the real optimum, the final hill climbing could come very near to the best solution, or even reach it, without getting stuck in too low local optima.

Perrier *et al.* [14] already proved a similar method to be promising, however they used a different complete algorithm and experimented in a different setting. We will discuss the differences in last section of the chapter.

6.2 Implementation

We used C++ for all the code; for the χ^2_{df} distribution and $\log \Gamma$ function implementations, needed for equations 4.8 and 4.1, we choose the boost library (http://www.boost.org/). In the next subsections we describe the implementation choices we made when they are different from papers or when they were not obvious.

6.2.1 MMPC first implementation

The first step was re-implementing the MMPC algorithm. We did it to be able to eventually modify every aspect of the procedure and to better integrate it with our method.

Fortunately in [3] the authors give a detailed description of all the algorithm used. The implementation of the pseudo-code we presented in chapter 5 is almost straightforward. Data sets are read from an input file and saved in a matrix of number of samples rows and a column for each variable. The output is organized as the input of B&B algorithm: the matrix of constraints obtained with the MMPC algorithm followed by the data set. Our implementation permits to modify all the parameters of the algorithm: the significance threshold α, the heuristic on the degree of freedom, the minimum number of samples to do the test, etc...

The only question to be answered was how to efficiently count all the $N_{ab\mathbf{c}}$ we need for the G-test. The problem is relatively easy using simple binary variables: we first use a simple example to explain the approach and then generalize it.

Suppose we need to test $T(x, y|z)$, where x, y, z are binary. We can consider the value of each variable as a digit of a binary number: for example $x = 0, y = 1, z = 1$ can be seen as 011, or 6 in decimal if we read the bits from left. The number of possible combinations, aka the number of counters we need, is in our case 8, furthermore we can easily give them an order. We can then store all our counter in a single array of int N with size 8, where every cell $N[i]$ represents the ith counter.

More clearly, $N[0]$ stores the total number of samples where $(x = 0, y = 0, z = 0)$, $N[1]$ the same for $(x = 1, y = 0, z = 0)$, $N[2]$ for $(x = 0, y = 1, z = 0)$, etc. We can then read the data set once and increment the right counter at each sample simply calculating the decimal number represented by the corresponding values of x, y, z.

The method is immediately generalized for every number of variables, but the difficulty is due the fact we can have variables that assume more then two values and especially variables with very different degrees of freedom in the same domain.

However, it is perfectly possible to order all the different patterns of values, and relative counters, in the same way we explained for the binary case.

First note that no matter the nature of the domain, we will always be working with discrete variables, and thus we can code the values of each variable with natural numbers, starting to 0; for example imagine we have variable w which can assume the "values" *child*, *adult* or *old*: in the data set child will correspond to 0, adult to 1 and old to 2.

Somehow we can then look at every value of every variable we in a certain sample as the digit of a number represented with a mixed base. Suppose to consider again x, y, z, where x and z binary and y can assume three values. First, the total number of counters will be $D(x)D(y)D(x) = 2 * 3 * 2 = 12$. The order of the combinations will be:

$$(x = 0, y = 0, z = 0) = 0$$

$$(x = 1, y = 0, z = 0) = 1$$

$$(x = 0, y = 1, z = 0) = 2$$

$$(x = 1, y = 1, z = 0) = 3$$

$$(x = 0, y = 2, z = 0) = 4$$

$$\ldots$$

$$(x = 1, y = 2, z = 1) = 11$$

We can calculate the "power" corresponding to each digit-variable multiplying the degrees of freedom; obtaining the digital number corresponding to a pattern or the opposite works exactly like the classical base-x/base-10 conversion. All the counters use the same logic.

6.2.2 Optimizing MMPC

We implemented all the optimizations described in [3] to improve the performance of MMPC'. We recall them here:

- Once a variable reaches a minimum association of zero with the target variable t, it is not considered again by the algorithm.
- Computations between subsequent calls to the Max-Min Heuristic are shared as follows. Suppose that in iteration i variable y is added to CPC of t, so that $CPC_{i+1} = CPC_i \cup \{y\}$, where the index denotes the iteration. The minimum association for any x in X with t conditioned on any subset of CPC_{i+1} can be written as

  $$\min\left(\min_{S \subseteq CPC_i} Assoc(x, t|S), \min_{S \subseteq CPC_i} Assoc(x, t|S \cup \{y\})\right)$$

  That is, the minimum over all subsets of CPC_{i+1} is the minimum between the minimum achieved with all subsets that do not include the new element Y and the minimum achieved with all subsets that include Y . In other words, only the newly created subsets by the addition of Y need to be tested for further minimizing the association.
- If a previous call $MMPC(x, D)$ has not returned t, then remove x from consideration during the invocation $MMPC(t, D)$: a subset $\mathbf{z}$ has already been found such that $Ind(x, t|\mathbf{z})$.
- If a previous call $MMPC(x, D)$ has returned t, then it is possible x also belongs in CPC of t . As a heuristic instead of starting with an empty CPC during the call $MMPC(t, D)$, include x in the starting CPC.

None of the optimizations alters the final output (and thus the correctness) of the algorithms but they significantly improve efficiency [3].

6.2.3 Permutation Tests

After the basic G-test we implemented a naive version the permutation independence test: we implemented the Algorithm 11 along with the **Rule 1**. To do it we needed to implement also the χ^2 test: the method used to calculate the G^2 metric was modified to include both the tests.

We were interested particularly in evaluating the precision of the methods and their interaction with the complete algorithm more then then optimization. We will discuss this choice and its implications with more detail along with the results.

6.2.4 Search Algorithms

For the search algorithm, we first decided to try a simple local search in the space of CPDAGs. We implemented the DAG-to-CPDAG and the PDAG-to-DAG algorithms we presented in chapter 3. The DAGs are represented as a simple binary matrix: a 1 at position (i, j) means there x_j is a parent of x_i. For PDAGs we need a slightly more complicated representation: we use the value 2 in the matrix to indicate the presence of an undirected edge, a -1 to indicate an incoming edge and a 1 to indicate a outgoing edge.

The AIC, BIC and BDeu local score functions described in section 4.3 were implemented; the counters needed are built from data every time the local score function is called, exactly like we did for the counters for the statistical tests (see 6.2.1).

The algorithm accepts in input a data set and a DAG G, then at each iteration do the following operations:

1. Obtain the CPDAG C from the DAG G using DAG-to-CPDAG
2. Apply on C all possible operators described in 4.3
3. Save the best structure G' obtained. If no improvement in the score was possible stops, if not go back to point 1

We point out that our implementation was very simple: to test the operators we blindly apply them, then check the validity using the algorithm PDAG-to-DAG; if a consist extension exists, we score it and confront it with the best structure found during the current iteration.

At this stage we were more interested in testing the effectiveness of the method which did not give good resutls and was dropped after the first preliminary experiments. In fact, a search using the space of CPDAGs after the B&B algorithm didn't give in many case any improvement in the score.

A simpler search in the space of DAGs however gave much better results. The final algorithm we implemented and used in the final step of the hybrid procedure is the simple greedy hill climbing presented in previous chapter (Algorithm 12), with the only difference that it can start from an existing structure G instead of the empty graph and constraints can be ignored.

6.3 Experiments

6.3.1 Setting

For our experiments we choose three networks used in [3]: CHILD (20 nodes), INSURANCE (27 nodes) and ALARM (37 nodes). We opted for the BDeu scoring function with $ESS = 1$ for all the experiments. We decide to set our experiments in a realistic scenario, thus we restricted the experiments on data set with small sample sizes and limited memory space: large sample sizes with respect to the number of variables are indeed rahter uncommon in real life problems [57].

Furthermore is important to note that, while the theorems found by de Campos and Ji decrease their efficiency when the sample size increases, making the B&B particularly suitable for our experiments, on the other hand the accuracy of traditional asymptotic independence tests decreases with low sample sizes. Comparing the different tests presented in [3] and in [13] is thus particularly interesting.

We consider for each network three different sample sizes (100, 200 and 500) and 10 data set for each size. The 500 sample data sets used are the ones available on-line at http://www.dsl-lab.org/supplements/mmhc_paper/mmhc_index.html. We obtained smaller data sets sampling randomly the data sets with 500 samples. We test the simple B&B algorithm, denoted with *NoC*, with two hybrid procedures using MMPC. The first adopt the the simple G-test and it will be denoted with *C06*. The second makes use of the permutation tests described in [13] (with 5000 permutations) and it will be denoted with *C10*. The statistical test used for algorithm 11 is the same used for C06.

We used the 32-bit de Campos program (version January 2012). The software permits to impose different restrictions; we only give a max to the memory (4000 MB). The version of the program we used has also a limit of 2^{30} iterations.

6.3.2 Score Results

Network	Samples	C06 - Score	C10 - Score	C06 - p	C10 - p
Child	100	99.84 ± 0.13	99.84 ± 0.14	**0.0156**	**0.0078**
	200	99.99 ± 0.02	99.99 ± 0.02	0.2500	0.5000
	500	99.99 ± 0.03	99.99 ± 0.04	0.2500	0.2500
Insurance	100	99.74 ± 0.14	99.61 ± 0.25	**0.0039**	**0.0039**
	200	99.85 ± 0.29	99.80 ± 0.29	**0.0078**	**0.0137**
	500	99.90 ± 0.14	99.76 ± 1.80	0.13	**0.0039**

Table 6.1: Comparison between the scores obtained with the B&B algorithm and the scores obtained with the two hybrid algorithms. Columns 3 and 4 report the ratio of the scores of the best solutions obtained by the C06 and C10 hybrid algorithms versus the NoC algorithm, for each network and sample size (percentages, *mean* $\pm$ *standard deviation* on 10 datasets). Columns 5 and 6 report the respective p-values of a Wilcoxon signed rank test comparing the results on the 10 data sets. Bold values correspond to p-values < 0.05.

In table 6.1 we present the comparison between the scores obtained with the two constrained algorithms (C06 and C10) and the unconstrained procedure (NoC) for each network and data size. It is immediate to verify that all the differences between the unconstrained and constrained score means are less then 0.5%. Considering the p-values returned by the Wilcoxon test with a significance threshold of 0.05, we note that for the Child Network the loss in score is significant only for the smallest instances. For Insurance, on the other hand, only the results for the C06 procedure with the largest sample size resulted not significantly different from the unconstrained procedure.

We point out that the algorithm reached the maximum number of iterations in all the instances for the Insurance network when no hybrid procedure was used: being the B&B algorithm an any-time procedure, we can expect the solutions in these cases to be approximate; however after trying a local search starting from the output DAGs, they resulted to be at least local optima or very close to local optima. We also must point out that, with the memory limit we imposed, the B&B algorithm was not able to build the cache for the unconstrained Alarm cases, thus the comparison between NoC and C06/C10 was not possible.

Network	Samples	C06	C10	p-value
Child	100	100	100	1
	200	100	100	1
	500	100	100	1
Insurance	100	100	99.87 ± 0.16	0,0781
	200	100	99.94 ± 0.18	0,3828
	500	100	99.85 ± 0.2	**0,0273**
Alarm	100	99.72 ± 1.09	100	0,3750
	200	99.38 ± 1.18	100	0,2324
	500	99.63 ± 0.6	100	0,0977

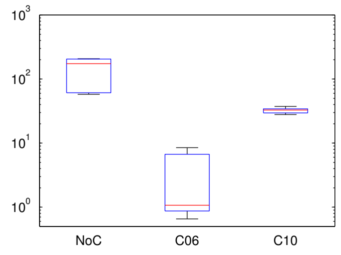
Table 6.2: Comparison between the scores obtained with the C06 and C10 hybrid algorithms. Columns 3 and 4 report the ratio of the scores obtained by the C06 and C10 hybrid algorithms versus the best of the two scores, for each network and sample size (percentages, *mean* $\pm$ *standard deviation* on 10 datasets). Column 5 reports the respective p-values of a Wilcoxon signed rank test comparing the results on the 10 data sets. Bold values correspond to p-values < 0.05.

To conclude, we compare more directly the scores obtained with the two different hybrid procedures. Looking at table 6.2 we can immediately see that only significantly different case is for the Insurance network with 500 sample size.

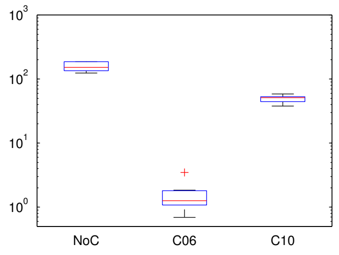
One could argue that the score is not the ultimate measure of how good a structure is; a different solution could be making a comparison between the structure found and the real network, if known, counting the number of missing, added or reverted edges. Score functions however are useful to easily evaluate results in the context of a specific data set, so every comparison we made is based on scores.

6.3.3 Time Results

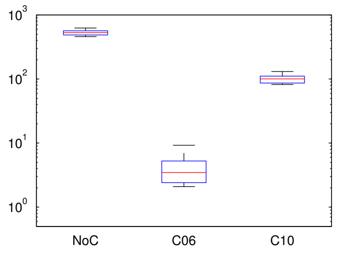
We now consider the time taken by the algorithm with the three different settings.



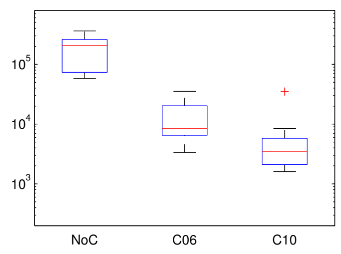
(a) Child 100



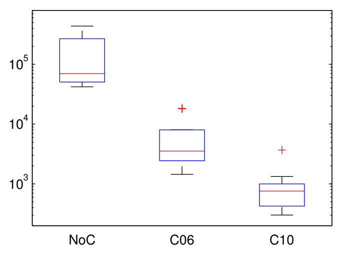
(b) Child 200



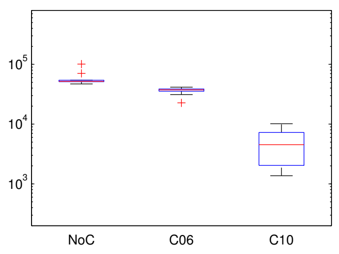
(c) Child 500



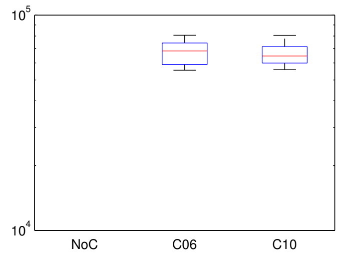
(d) Insurance 100



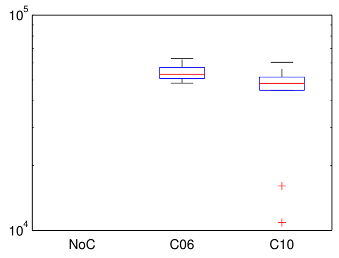
(e) Insurance 200



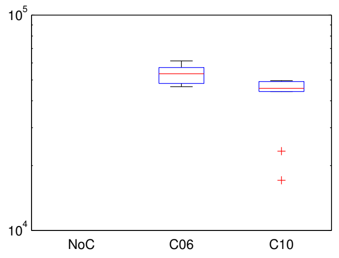
(f) Insurance 500



(g) Alarm 100



(h) Alarm 200



(i) Alarm 500

Figure 6.2: Box plots of the total time needed by the NoC, C06 and C10 procedures for the Child (first row), Insurance (second row) and Alarm (third row) networks and for sample sizes equal to 100 (first column), 200 (second column) and 500 (third column). For NoC, the execution time of the B&B algorithm is reported. For C06 and C10, the sum of the execution times of the three phases of the hybrid procedures is reported.

In every case the hybrid procedures took a time considerably smaller, between one and two orders of magnitude. The more precise constraints given by the permutation tests (C10) proved their usefulness and the total time for Insurance cases is much smaller than the simple asymptotic tests (C06, all p-values obtained with Wilcoxon signed rank test < 0.0039).

For the Child Network, however, it is exactly the opposite: this is probably due the fact that the time for calculating the constraints themselves is much longer for the permutation cases, while the B&B phase required very small time for both C06 and C10(all p-values < 0.002).

In the Alarm case the total running times are much more similar, but almost all computations reached the maximum number of iterations. It's worth mentioning, however, that in a couple of instances for both 200 and 500 samples the C10 hybrid procedure led to less then 2^{30} iterations for the B&B algorithm and thus to much smaller times, which are clearly visible as outliers in the box plots. Differences in execution time, however, increase with the sample size and are significant for the datasets with 500 samples (p-value < 0.0195).

6.3.4 Search Space Results

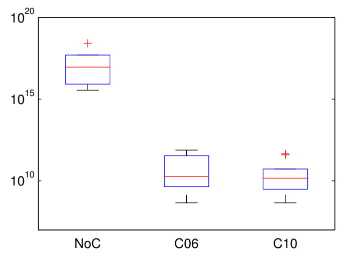
We now make some considerations on the sizes of the search space. We summarize our results in figure 6.3.

We first point out that the sizes of original search spaces, which are not reported in the box plots, is several order of magnitude bigger than the one obtained with the simple NoC procedure, i.e. with the application of the theorems for the cache reduction without any further constraints. To be precise, the original search space reported by the B&B algorithm was $2,46e + 114$ for Child and $2,10e + 211$ for Insurance. For Alarm the value reported was *inf*, i.e. it was greater then the maximum representation for the 32-bit program.

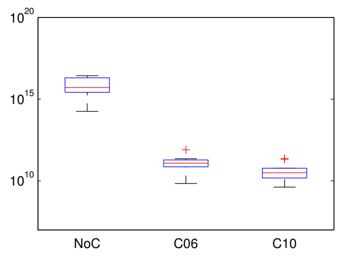
It is immediate to verify that the constraints imposed with C06 and C10 lead to much smaller search spaces. This was to be expected, but we note that the difference, in therms of orders of magnitude, is even more pronounced than analysing the computational time. We note also that the C10 procedure is in turn sensibly better then the C06, particularly for the larger networks and sample sizes.

Considering Insurance and Child networks cases and observing the behaviour of NoC algorithm, we can note how there is a sensible worsening on the effectiveness of de Campos and Ji theorems for the larger sample sizes. However, we can also note that the constraints used by C10 and C06 procedure somehow balance this situation; this is particularly clear in the Alarm case.

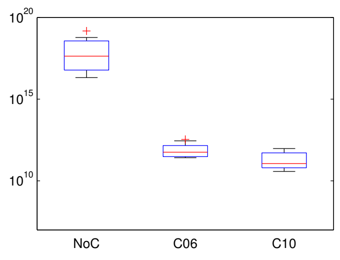
We stress that the reduced space for C10 and C06 is the combination of



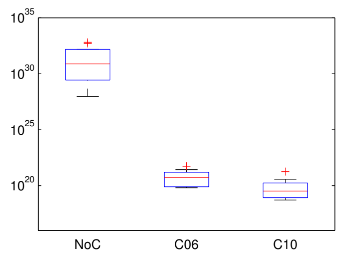
(a) Child 100



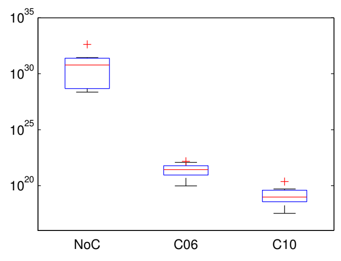
(b) Child 200



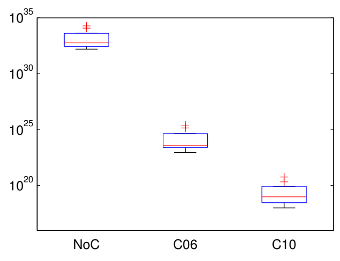
(c) Child 500



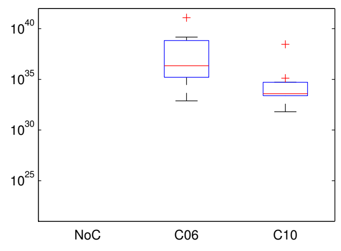
(d) Insurance 100



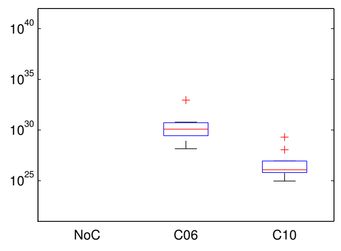
(e) Insurance 200



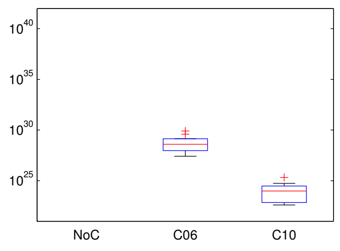
(f) Insurance 500



(g) Alarm 100



(h) Alarm 200



(i) Alarm 500

Figure 6.3: Box plots of the sizes of the search space, returned in the B&B output, after the construction of the cache, and thus the applications of the mathematical criterion on BDeu score function.

the mathematical theorems used in the cache construction phase by the B&B algorithm and the constraints introduced with the MMPC algorithm. We point out that, observing the outputs, we could verify that, if the MMPC found, for example, m variables in the CPC for variable x, the number of cache entries, i.e. combinations of parents for x, is much smaller then the total number of possible combinations, which should be 2^m.

6.3.5 Comparison with related works and discussion

In this chapter, we presented a performance analysis of two hybrid algorithms for Bayesian Networks structure learning. The two algorithms were obtained hybridizing a complete state-of-the-art method with the MMPC algorithm, using two different statistical test: a simple asymptotic test and a permutation test.

A similar approach was tested in 2008 by Perrier *et al.* [14]; they experimented the use of the MMPC algorithm to obtain constraints for a complete algorithm based on dynamic programming, followed by local search. The work demonstrated the potentiality of statistical constraints applied to a complete algorithm and the authors provided an interesting comparison between different algorithms on random networks. However, the size of the networks used for the comparisons by Perrier *et al.* was limited to 20 nodes. Moreover, the data sets used were very large (from 500 up to 10000 samples) and no limits were imposed to available computational resources.

Instead, we used a recent state-of-the-art complete algorithm due de Campos and Ji and we set our experiments in a realistic scenario: we imposed e reasonable memory limit and we used data set with fewer samples. This is together a necessity and a challenge. On the one hand the proprieties found by de Campos and Ji degrade with the increase number of data [5], which however tend to be rare in real life problems. On the other hand independence tests are less precise on scarce data.

Both the independence tests originally used for MMPC (C06) and the more recently proposed ones (C10) based on permutations of data have been implemented to verify pros and cons; in fact in the context of a complete algorithm followed by an unconstrained local search the trade-off between computational time, quality of result and memory is not obvious and can't be deduced directly from precedent work, especially in our restricted scenario.

Our results are promising and may help to clarify the subject: both tests improved dramatically the computational time against a limited loss in score. The hybrid procedures also permitted to learn a larger network, Alarm, for which the normal B&B algorithm failed due to the memory limit. The permutation tests demonstrated to be more more precise in a sensible way, thus further reducing the time required by the B&B algorithm, but due to their longer calculation times they are probably not convenient for small networks and the B&B algorithm.

Chapter 7

Conclusions and Future Work

We described Bayesian Networks, graphical models that represent conditional independence properties over a set of random variables in a compact and intuitive way. Since uncertainty appears to be an inescapable aspect of most real-world applications, Bayesian Networks have seen a growing popularity in the last few years.

We outlined the general approaches that can be found in literature about the complex problem of building the graph of a network itself. A first approach makes use of statistical tests of independence in order to retrieve the skeleton of the network and then try to recover the direction of the edges [2]. A second approach makes use of scoring functions to asses the quality of a structure, and then explores the search space; typically the search space consists in all possible DAGs or CPDAGs, i.e. Bayesian Network equivalence classes. We gave the details of AIC, BIC and BD score metrics and some examples of search algorithms; both heuristic and complete methods can be found in literature [2].

We described in detail the MMHC algorithm due to Tsamardinos *et al.*[13] which can be considered an hybrid between the search & score and the statistical test based approaches. The algorithm first build a set of possible parents or children for each variable using statistical tests of independence following the so called MMPC algorithm, then use a simple hill climbing greedy search algorithm in the space of DAGs constrained to the sets found.

Recently Tsamardinos *et al.* [13] proposed the use of permutation tests of independence for data set with small sample sizes.

We described also a recent complete algorithm due to de Campos and Ji [5] which presents an approach different than dynamic programming algorithms and can be considered the state-of-the-art for complete methods.

The work of this thesis consists in an hybrid approach realized using the MMPC algorithm as structural constraints for the complete B&B method, followed by a simple local search in the total search space to recover edges eventually excluded by statistical tests in the first phase. We made experiments for three well know networks in a realistic scenario: we used small sample sizes data sets and imposed a reasonable memory limit to the B&B algorithm. We compared the results obtained with both simple asymptotic tests and the more recently proposed permutation tests of independences with the simple unconstrained B&B algorithm.

In 2008 Perrier *et al.* adopted in [14] a similar approach; however in [14] the comparison between the different algorithms is limited to 20 nodes, while we used a more recent complete algorithm and thus we could make comparisons for a bigger network (27 nodes). Moreover Perrier *et al.* used much bigger sample sizes (from 500 to 10000) and no limits for computational resources, while we set our experiments in a much more realistic scenario.

The main conclusion is that our results are promising and show that the statistical constraints, even with limited sample sizes, bring a minimal loss in term of score, while the gain in computational time is considerable. In some cases the results are demonstrated to be indistinguishable by Wilcoxon signed rank test. The hybrid procedures also permitted to learn a larger network, Alarm, for which the normal B&B algorithm failed due to the memory limit we imposed.

A possible future direction of our work will be a systematic comparison between our hybrid procedures and meta-heuristic methods exploiting the MMPC constraints, in the context of small sample sizes. A simple algorithm that could be used is the classic MMHC [3], or its variant, as the one proposed in [14], where a first hill climbing search, limited to the hypothetical skeleton found by MMPC, is followed by a second local search without any constraint.

The problem with heuristic methods is that they tend to get stuck on local optima; restricting the search space with MMPC can have beneficial effects, but small sample sizes, as we discussed, are not ideal. An approach we would like to attempt is re-iterated hill climbing: the algorithm would start from the empty graph, reach the first solution, then perturb it and start the search again, repeating the process for a certain number of times or until some condition is met.

This is not a new approach per se [8], but using it with the two-phases hill climbing as in [14] could be interesting. The use of CPDAGs, which did not give good results for the final local search of our hybrid procedure, in this context could potentially give some benefit, considering that they represents equivalence classes of Bayesian Networks, thus a smaller search space.

Another interesting direction will be to study the effect of variations of the significance threshold used by the independence tests of the MMPC algorithm, now set to a default value of 0.05, on the overall performance of the hybrid algorithms.

Furthermore, a variation on the hybrid algorithm that could be worth investigating consists in exploiting an other characteristic of the B&B algorithm. In fact, it is possible to give a starting network in input in order to improve the computation. An idea would be to obtain a first solution as we did for our experiments, then use the solution in input along with constraints obtained with a more generous statistical test. The objective again would be to obtain the best network, or an acceptable approximation, in less time and/or memory.

Such procedure, however, to be efficient, should be able to reuse the cache calculated for the first run of the algorithm and simply expand it, which is not possible with the current software available.

Acknowledgements

First and foremost I offer my sincere gratitude to my supervisors, Prof Silvana Badaloni and Dr Francesco Sambo, for the constant and friendly help and for the trust given to me. A special thanks to Cassio P. de Campos for his constant feedback and precious advices on his software tool.

I would like to tank all the friends I have met in University, especially Federica and the entire "company of lunch time" for the right words at the right time, and Davide and Nicola for the great days in Barcelona last October. Paolo offered his patience in endless exams we studied together, without him all would have been much less fun and much more difficult.

Thanks also to all the friends that saved me outside University. A special mention to Anna Miriam, for all the midnight chatters and her common sense, and to Chiara: without her, life would be way too boring, and the world a much sadder place. Thanks to Jacopo, Matteo, Marco, Arianna, Melissa and Maristella for the last months together (and for Arkham Horror !).

The memories of the time spent in Grenoble are extraordinarily precious to me, thus I take this opportunity to express also my gratitude to the European Union for the Erasmus program, and to remember all the wonderful people I had the opportunity to meet in France, *merci à tous*!

Finally, I would like to thank my family for the support throughout all this time.

Bibliography

[1] D.Koller and N.Friedman. *Probabilistic Graphical Models. Principles and Techniques.* The MIT Press, Cambridge Ma, London England, 2009.

[2] Rónán Daly, Qiang Shen, and Stuart Aitken. Learning bayesian networks: approaches and issues. *The Knowledge Engineering Review*, pages 220–227, 2011.

[3] I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max min hill climbing bayesian network structure learning algorithm. *Machine Learning*, (65):31–78, 2006.

[4] Cassio P. de Campos, Qiang Ji, and Zhi Zeng. Structure learning of bayesian networks using constraints. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

[5] Cassio P. de Campos and Qiang Ji. Efficient structural learning of bayesian networks using constraints. *Journal of Machine Learning Research*, (12):663–689, 2011.

[6] Hirotugu Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, (19):716–723, 1974.

[7] Gideon Schwarz. Estimating the dimension of a model. *Annals of Statistics*, (2):461–464, 1978.

[8] David Heckerman, Dan Geiger, and David M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, (20):197–243, 1995.

[9] David M. Chickering. Learning bayesian networks is np-complete. *Learning from Data: Artificial Intelligence and Statistics*, page 121–130, 1996.

[10] R. Robinson. *Counting labelled acyclic digraphs.* Academic Press, 1973.

[11] P. Spirtes and C. Glymour. An algorithm for fast recovery of sparse causal graphs. *Carnegie Mellon University*, 1990.

[12] David M. Chickering. Learning equivalence classes of bayesian-network structures. *Journal of Machine Learning Research*, (2):445–498, 2002.

[13] I. Tsamardinos and G. Borboudakis. Permutation testing improves bayesian network learning. *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, page 322–337, 2010.

[14] Eric Perrier, Seiya Imoto, and Satoru Miyano. Finding optimal bayesian network given a super-structure. *Journal of Machine Learning Research*, (9):2251–2286, 2008.

[15] C. M. Bishop. *Pattern Recognition and Machine Learning.* 2006.

[16] B. Fristedt and L. Gray. *A modern approach to probability theory.* Boston: Birkhäuser, 1996.

[17] Russell J. Stuart and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2 edition, 2003.

[18] Spirtes, Glymour, and Scheines. *Causation, Prediction and Search.* MIT Press, 2 edition, 2000.

[19] Spirtes, Glymour, and Scheines. *Causation, Prediction and Search.* Springer/Verlag, 1 edition, 1993.

[20] R. Neapolitan. *Learning Bayesian networks.* Prentice Hall, 2003.

[21] O. Pourret, P. Naim, and B. Marcot. *Bayesian Networks: A Practical Guide to Applications.* Statistics in Practice Wiley, 2008.

[22] U. B. Kjaerulff and A. L. Madsen. *Bayesian networks and influence diagrams: a guide to construction and analysis.* Springer, 2008.

[23] S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjrul, M. Woldbye, A. R. Srensen, A. Rosenfalck, and F. Jensen. Munin–an expert emg assistanta. *Computer-aided Electromyography and Expert Systems*, page 255–277, 1989.

[24] P. J. F. Lucas, L. C. van der Gaag, and A. Abu-Hanna. Bayesian networks in biomedicine and healthcare. *Artificial Intelligence in Medicine*, (30):201–214, 2004.

[25] I. Beinlich, H. Suermondt, R. Chavez, and G. Cooper. The alarm monitoring system: a case study with two probabilistic inference techniques for belief networks. In *Proceedings of the Second European Conference on Artificial Intelligence in Medicine*, 1989.

[26] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, (29):131–163, 1997.

[27] B. Abramson, J. Brown, W. Edwards, A. Murphy, and R. L. Winkler. Hailfinder: a bayesian system for forecasting severe weather. *International Journal of Forecasting*, (12):57–71, 1996.

[28] J. Forbes, T. Huang, K. Kanazawa, and S. Russell. The batmobile: towards a bayesian automated taxi. In *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, page 1878–1885, 1995.

[29] N. Friedman. Inferring cellular networks using probabilistic graphical models. *Science*, (303):799–805, 2004.

[30] T. Verma and J. Pearl. Equivalence and synthesis of causal models. *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 220–227, 1990.

[31] David M. Chickering. Learning equivalence classes of bayesian-network structures. In *Proceedings of the Twelfths Conference on Uncertainty in Artificial Intelligence*, pages 150–157, 1996.

[32] C. Meek. Causal inference and causal explanation with background knowledge. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 403–410, 1995.

[33] S. A. Andersson, D. Madigan, and M. D. Perlman. A characterization of markov equivalence classes for acyclic digraphs. *Annals of Statistics*, (25):505–541, 1997.

[34] D. M. Chickering. A transformational characterization of bayesian networks structures. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 87–98, 1995.

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

[36] R. A. Fisher. *Statistical Methods for Research Workers.* Edinburgh: Oliver and Boyd, 1925.

[37] Duncan Cramer and Dennis Howitt. *The Sage Dictionary of Statistics.* 2004.

[38] A. Agresti. *Categorical Data Analysis.* Wiley-Interscience, Hoboken, 2 edition, 2002.

[39] G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, (9):309–347, 1992.

[40] E. Herskovits and G. Cooper. Kutatò : an entropy-driven system for construction of probabilistic expert systems from data. In *Uncertainty in Artificial Intelligence 6*, page 54–62, 1991.

[41] R. R. Bouckaert. Probabilistic network construction using the minimum description length principle. In *Symbolic and Quantitative Approaches to Reasoning and Uncertainty: European Conference ECSQARU '93*, page 41–48, 1993.

[42] A. L. de Santana, C. R. Frances, C. A. Rocha, S. Carvalho, N. L. Vijaykumar, L. P. Rego, and J. C. Costa. Strategies for improving the modelling and interpretability of bayesian networks. *Data and Knowledge Engineering*, (63):91–107, 2007.

[43] F. Liu and Q. Zhu. The max-relevance and min-redundancy greedy bayesian network learning algorithm. In *Bio-inspired Modeling of Cognitive Tasks:*

Proceedings of the Second International Work - Conference on the Interplay between Natural and Artificial Computation, page 346–356, 2007.

[44] D. M. Chickering, D. Geiger, and D. Heckerman. Learning bayesian networks: search methods and experimental results. In *Learning from Data: Artificial Intelligence and Statistics*, page 346–356, 1996.

[45] H. Steck. On the use of skeletons when learning in bayesian networks. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, page 558–565, 2000.

[46] K. Kim and S. Cho. Evolutionary aggregation and refinement of bayesian networks. In *Proceedings of the IEEE Congress on Evolutionary Computation*, page 1513–1520, 2006.

[47] L. M. de Campos and J. F. Huete. Approximating causal orderings for bayesian networks using genetic algorithms and simulated annealing. In *Proceedings of the Eight Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, page 1333–340, 2000.

[48] H. Xing-Chen, Q. Z. T. Lei, and S. Li-Ping. Learning bayesian network structures with discrete particle swarm optimization algorithm. In *Proceedings of the IEEE Symposium on Foundations of Computational Intelligence*, page 47–52, 2007.

[49] R. Daly and Q. Shen. Learning bayesian equivalent classes with ant colony optimization. *Journal of Artificial Intelligence Research*, (35):391–447, 2009.

[50] Tomi Silander and Petri Myllymaki. A simple approach for finding the globally optimal bayesian network structure. In *Conference on Uncertainty in Artificial Intelligence, pages*, page 445–452, 2006.

[51] M.Koivisto and K. Sood. Exact bayesian structure discovery in bayesian networks. *Journal of Machine Learning Research*, (5):549–573, 2004.

[52] D. Eaton and K. Murphy. Bayesian structure learning using dynamic programming and mcmc. In *Proceedings of the Twenty-third Annual Conference on Uncertainty in Artificial Intelligence*, page 101–108, 2007.

[53] Nir Friedman, Iftach Nachman, and Dana Pe´er. Learning bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Fifteenth Conference on Uncertainty in Artificial Intelligence*, 1999.

[54] P. Good. *Permutation, Parametric, and Bootstrap Tests of Hypotheses*. Springer, Heidelberg, 3 edition, 2004.

[55] T. Jaakkola, D. Sontag, A. Globerson, and M. Meila. Learning bayesian network structure using lp relaxations. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, page 358–365, 2010.

[56] M. Teyssier and D. Koller. Ordering-based search: A simple and effective algorithm for learning bayesian networks. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, page 584–590, 2005.

[57] P. Sebastiani, M.M. Abad, and M. F. Ramoni. Bayesian networks for genomic analysis, 2004.