Università degli Studi "Roma Tre"

Scuola dottorale in "Economia e metodi quantitativi"

XXIII Ciclo

STRUCTURAL LEARNING OF BAYESIAN NETWORKs FROM ORDINAL DATA

Flaminia Musella

A.A. 2010/2011

Supervisor:	*Coordinatore:*
Prof. Paola Vicard	Prof. Julia Mortera

To Massimiliano

Abstract

In observational studies many features are measured on a sample in a given time. When the measurement scale is ordinal, observed variables are categorical ordinal variables. Their increasing presence in databases has influenced the development of methods for ordinal data analysis (Joe 1971; Clogg and Shihadeh 1994; Agresti 2010). Frequently, researchers are interested in the multivariate analysis and dependencies (Cox and Wermuth 1996). Graphical models (Lauritzen 1996) can be useful for this purpose: they are a family of multivariate statistical models that study dependencies among variables and provide a representation of them by means of graphs. Among these models, Bayesian networks (Cowell *et al.* 1999) represent the joint distribution of a set of variables using directed acyclic graphs (DAGs). When the structure of phenomenon is unknown (or partially known), building the DAG manually may be difficult, but the network can be learnt directly from data. This phase, called structural learning, can be performed following different approaches. However, there are few methods suitable for ordinal data. The main task of this PhD thesis is to perform the structural learning of Bayesian networks in presence of ordinal variables. As original aspect, a new procedure able to take into account information provided by ordinal variables, has been developed. The new algorithm, called OPC, represents a variation of one of the most used and well-known constraint-based algorithms, namely PC (Spirtes *et al.* 2000). A nonparametric test, appropriate for ordinal variables, has been used in the OPC procedure. Some simulation studies have been conducted in order to evaluate and compare the performance of PC and OPC algorithms. On the basis of results, the main features and limitations of the OPC procedure are discussed.

Acknowledgements

This thesis contains the research I have conducted during PhD course of studies in “Metodi Statistici per l’Economia e l’Impresa”[†]. First of all I would like to thank Prof. Julia Mortera, who coordinates the PhD activities, and all professors that, with their courses and seminars, have contributed to my training over these years. I desire to warmly acknowledge Prof. Paola Vicard: she introduced me to the area of graphical models and she patiently supervised my work. I have appreciated very much her willingness.
I am also grateful to Department of Economics, Roma Tre University, and PRIN2007 project of MIUR[*] for having given me the opportunity to attend extra courses and to take part to some conferences both in Italy and abroad. I am particularly grateful for having financed me during my stay in Oxorfd, UK. About this, I am deeply indebted to Prof Steffen Lauritzen who gave me the opportunity to study at the Department of Statistics, Oxford University, for four months. He involved me in writing the R code with Prof. David Edwards who is another professor I desire to warmly thank.
I am also grateful to Prof. Anna Gottard for the time she dedicated me at Warwik University - UK.
A special thank to Prof. Pierluigi Conti and Prof. Alan Agresti that suggested to me some relevant references. Many thanks to Dr. Roberto Di Manno who helped me getting started.
I would also like to thank Prof. M. Francesca Renzi who involved me in some

[†]Statistical Methods for Economics and Enterprise

[*]Italian Ministry of Education, University and Research

applied research projects in these years.
Many thanks to my colleagues that have shared with me this experience and to my friends (Chiara, Emanuele, Francesca, Roberta) that have shared with me the stress.
My deepest thank to my husband and to my family that have always supported and encouraged me during these three years.

Contents

Chapter 1

Introduction

This thesis deals with structural learning of Bayesian networks from categorical ordinal data. The aim is to propose a constraint-based procedure that is a variation of PC algorithm. The advantage of our algorithm is that it is appropriate for ordinal data. For this reason, we start providing preliminaries on categorical variables and, in particular, on ordinal variables.

1.1 Categorical variables

Statistical variables can be classified in different ways according their nature (*qualitative* or *quantitative* variables), according to the set of values over which the variable is defined (*continuous* or *discrete* variables), according to the measurement scale (*nominal*, *ordinal* or *ratio*). We are interested in those variables whose measurement scale is based on a set of categories and that are called *categorical variables*. These variables are common in several research areas such as social science, biostatistics, genetics, education and marketing. The increasing presence of them in datasets has stimulated the development of appropriate methods for analysing categorical variables (Agresti 2002). Many types of variables can be labelled as categorical; they are:

- *nominal* variables: these take distinct values on a qualitative scale. The categories of nominal variables do not have an ordering. Some examples are gender (that takes categories "male" or "female"), eye color ("blue", "brown", "green", "gray", "other"), favorite type of residence ("house", "flat", "tower", "other"). Levels of nominal variables are qualitatively different but not quantitatively since they are a list of nouns.

- *ordinal* variables: levels of such variables are clearly and naturally ordered but distance among categories cannot be established. Ordinal scales are used in many situations. For instance, can be used for measuring attitudes and opinions such as satisfaction degree (that can take levels "very dissatisfied", "little dissatisfied", "moderately satisfied", "very satisfied"), for expressing stages of a disease ("first", "second", "third"), for classifying levels of education ("none", "high school", "bachelor's", "master's degree", "doctorate"). Number of levels depend on what we are measuring and how we decide to measure. Scales can have an odds or even number of categories. Also *discrete* variables having few levels and *continuous* variables collapsed in a small number of ordered classes can be considered as ordinal. Some examples are the number of cars in a family (0,1,2) and the annual incoming measured on a continuous scale and summarised in distinct ordered classes.

The type of variable determines which statistical method is more appropriate in the data analysis. Nevertheless, since ordinal variables are hierarchically higher than nominal, some techniques for nominal variables are commonly used with ordinal data. Following this method, the ordering among categories is not considered. Ignoring such ordering produces a (sometimes relevant) loss of information. Furthermore, results gained with nominal methods may be quite different from those achieved using ordinal variables (Clogg and Shihadeh 1994; Agresti 2002). In detail, Agresti (2010) discusses some advantages coming from treating ordinal variables as ordinal rather than nominal. The most relevant are:

- ordinal models can be more parsimonious and can be simpler to adopt and interpret;
- ordinal analysis can give more powerful results.

The last consideration is essentially the reason why we are interested in preserving the ordering also in graphical modelling. As it will be clear in Chapter 2, graphical models are multivariate statistical models for studying independence between variables. Methods for testing independence between ordinal variables can be more powerful than those between nominals. However, there are few ordinal sensitive procedures for learning Bayesian networks structure from ordinal data. So, the motivation of this dissertation is to propose an algorithm that takes into account information provided by ordinal variables. The notion of association for ordered categories is a key concept for our purpose and is introduced in the following Section.

1.2 Association for ordinal variables

Relationships between categorical variables are commonly displayed by means of tables. Let X and Y be two categorical variables with T and C categories respectively. It is possible to classify units on both variables by a table with T rows and C columns. The cells of the table represent all possible outcomes and contain frequency counts. Such a table is called *contingency table* or *cross-classification table* or $T \times C$ table (see Table 1.1). Using the common notation, the generic cell (i, j) stands for the pair with response $(X = i; Y = j)$; the frequency count of outcome $(X = i; Y = j)$ is denoted by n_{ij}.

	Y				
X	1	2	....	C	Total
1	n_{11}	n_{12}	....	n_{1C}	n_{1+}
2	n_{21}	n_{22}	....	n_{2C}	n_{2+}
....	....	....	....	....	....
T	n_{T1}	n_{T2}	....	n_{TC}	n_{T+}
Total	n_{+1}	n_{+2}	....	n_{+C}	n_{++}

Table 1.1: A generic contingency table

The table can also display the probability distribution π_{ij}, that is the joint distribution of $(X = i; Y = j)$, and marginal distributions of X and Y, that are respectively denoted by $\pi_{i+} = \sum_j \pi_{ij}$ and $\pi_{+j} = \sum_i \pi_{ij}$. Two variables, X and Y are said to be *independent* if $\pi_{ij} = \pi_{i+}\pi_{+j}$, for all i and all j.

Given a contingency table, we can summarise the association between variables by a single index. Measures of association for categorical variables are largely discussed in the literature (Agresti 2002, e.g.). In this Section, we recall some measures of association for categorical ordinal variables only.

Categories of ordinal variables have an inherent ordering that provides an additional information beyond that possessed by nominal variables. For this reason some appropriate measures of association have been developed (Agresti 2010). In particular, in presence of ordinal variables it is common looking for a monotone trend. Generally speaking the monotone association is the tendency of Y to increase/decrease as X does. So, it is frequent to deal with the concept of positive dependence (or concordance) and negative dependence (or discordance). In detail, a pair of observations is *concordant* if the the subject ranked higher on X, ranks higher on Y too. A pair is *discordant* if the subject ranked higher on X, ranks lower on Y. A pair is *tied* if subjects rank the same on both variables.

Several measures are based on the number of concordant and discordant pairs of observations. Some of the most used measures are:

- *Gamma* (Goodman and Kruskal 1979): is a symmetric measure denoted by γ and varying between -1 and 1. It takes positive values when variables are positive associated; it is negative if the association is negative. Independence implies that $\gamma = 0$ but the converse is not true;

- *Yule's Q* (Yule 1912): this measure is the *Gamma* index computed on a 2×2 table;

- *Kendall's tau-b* (Kendall 1945): is a symmetric measure less sensitive than γ; it is mainly based on the difference between concordant and discordant pairs. It assumes values between -1 and 1 but it reaches $|1|$ only for square tables. It is equal to 0 when variables are independent.

Here, we only use *Gamma* as measure of monotone association. Formula of *Gamma* index and an illustrative example is available in Section 3.4.
Having introduced the key concepts of ordinal variables and association for ordinal variables, a summary of the aim of this dissertation and an overview of the thesis are provided in the following Sections.

1.3 Goals and contribution

This dissertation deals with Bayesian networks modelling. A Bayesian network is a graphical model that provides a representation of independence structure between variables by a Directed Acyclic Graph (DAG). When the dependence structure of phenomenon is known, the graph can be manually built on the basis of expert knowledge; if the structure is unknown, the network can be automatically learnt from data. The last case is largely studied and discussed in the literature (Neapolitan 2003, e.g.) and it is known as structural learning. The main approaches to structural learning are the *scoring and searching* and the *constraint-based*. The first is based on a searching, in a specific space, of the model that has the best score given a chosen metric; the second focuses on the conditional independence relations revealed from the data. More specifically, algorithms belonging to the constrain-based approach carry out a sequence of independence statistical tests and draw the network in according to the test results. Even if the field is quite mature, this work contributes by introducing a new procedure that is appropriate for ordinal data. The technique is an attempt to learn a network in presence of ordinal variables without demoting them in nominal. More specifically, the focus is on a new constraint-based algorithm capable to learn a DAG structure from ordinal data; the procedure involves a nonparametric test for monotonic trend. The new algorithm, namely OPC (Ordinal PC) algorithm, represents a variation of one of the most used algorithms based on independence test and called PC algorithm. Applications here presented show that, when ordinal variables occur and the sample size is small, the new algorithm is a more efficient solution than PC algorithm.

1.3.1 Purpose and objectives

The purpose of this research is thus two-fold:

1. to develop a method for learning Bayesian networks structure without ignoring the additional information provided by ordinal variables.
   This is achieved by replacing the test used by PC algorithm and computed on nominal-nominal tables with a nonparametric test on ordinal-ordinal tables.

2. to test the structural accuracy of OPC algorithm in comparison with that of PC algorithm.
   Simulation studies have been conducted with the aim to measure and compare algorithms performance.

A further objective is to introduce another algorithm called NOPC (Nominal-Ordinal PC). This represents a natural extension of OPC algorithm. The procedure is appropriate in presence of mixed (nominal and ordinal) variables. It includes some nonparametric tests that can be used to check conditional independence for categorical data. The NOPC algorithm is presented and discussed but its performance are not tested in this dissertation.

1.3.2 Data and software

Simulations have been conducted using two different datasets.

- *Customer Satisfaction data*: these data represents a portion of a dataset coming from a real survey of customer satisfaction about services delivered by an Italian post-office; the sample is made of 228 units that have been interviewed by means of a face-to-face questionnaire; variables have been measured on three ordinal levels and concern the following six aspects: reliability, reassurance capacity, tangible aspect, empathy, response capacity and overall satisfaction.

- *Political Action data*: these data, well-known in literature (Barnes and Kaase 1979), come from a cross-national survey conducted with the aim to measure attitude and opinions on politics. Data are a selection of six variables measured on a six-point ordinal scale and linked to the concept of political efficacy: NoSay, voting, complex, NoCare, touch, interest. Only records without missing values (768) have been taken into account.

Software used for implementing the algorithm is `R` (R Development Core Team 2009). Some new functions have been written and some already existing commands have been used. `R` packages used, additional to those standard, are:

- `pcalg` (Kalisch and Bühlmann 2007) for structural learning algorithms. In detail the package provides a version of PC algorithm by `pcAlgo` function. This has represented the starting point for developing both PC and OPC function.
- `RHugin` (Konis and Expert. 2010) for generating several datasets according to a DAG;
- `igraph` (Csárdi and Nepusz 2006) for plots.

1.4 Thesis overview

The emphasis of this dissertation is to learn a Bayesian network when ordinal variables occur. The innovative contribution is given by the OPC algorithm, that is a constraint-based algorithm developed for handling the structural learning from ordinal data. The thesis is organized in the following chapters:

Chapter 2 introduces some basic concepts of graph theory and conditional independence. It provides an introduction on independence graphs as a tool for summurising the interactions within a set of variables; Markov properties are discussed as the formal link for merging probabilistic and graphical aspects in graphical models. Finally, it deals with Bayesian networks and their main features.

Chapter 3 providing the main elements about structural learning and, more specifically, about constraint-based algorithms, constitutes the core of the thesis. The central topic is the PC algorithm and its procedure: each step of the algorithm is analysed and the main limits are discussed. This chapter also describes some nonparametric tests that are relevant for checking conditional independence between discrete variables. The main focus is on the Jonckheere-Terpstra test, appropriate for ordinal variables, that is the tool used by OPC algorithm. This chapter also describes the NOPC algorithm.

Chapter 4 introduces some performance indicators for comparing algorithms. More specifically, the aim of this chapter is to compare performance of algorithms. It describes the experimental studies developed on 1000 samples simulated according to two different networks. Results are presented with respect to different sample sizes (50, 100, 500).

Conclusion and discussion summurises the main research findings and discusses the relevance of this research. It also highlights the further developments related to the current work.

Appendices provide codes of functions written in `R`. In detail, Appendix A includes a suite of `R` functions among which OPC and NOPC; in Appendix B there is the `R` code used for carrying out simulations.

Chapter 2

Basics on Bayesian networks

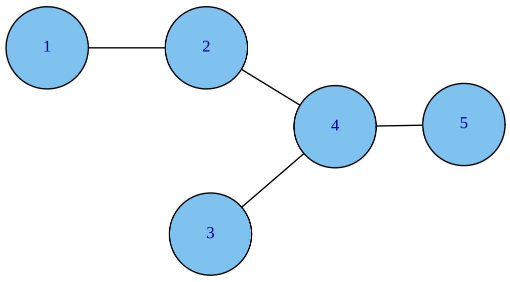
Graphical models are multivariate statistical models that provide a pictorial representation of probabilistic distributions. Diagrams used for displaying the statistical model are graphs. The origins of graphical models can be traced back to the beginning of 1900; the first applications were in physics (Gibbs 1902) and genetics (Wright 1921) but they have been applied in several fields such as economics (Wold 1954) and social science (Blalock 1971). More recently, graphical models have been discussed by Whittaker (1990), Lauritzen (1996) and Cox and Wermuth (1996).
The large applicability of graphical models is mainly due to their relevant features; one of the main advantages is that they can be applied for modelling discrete, continuous and mixed variables. Furthermore, the appealing graphical representation of model provides a natural and intuitive way to read off the independences statements between variables. As a consequence of this, graphical models make easy communication between experts of a specific domain and statisticians. Another relevant aspect of some graphical models is their ability to combine small problems in larger problem. This feature, called modularity, makes the tool powerful to handle complex statistical models involving many variables.
In recent years, the interest in graphical models increased with the use of Bayesian networks in *expert systems* (Cowell *et al.* 1999). Bayesian networks are a particular kind of graphical models based on directed acyclic graphs.

Probabilistic expert systems combine Bayesian networks and computational aspects with the aim to encode and summarise specialist expertises in a tool approachable also by non experts.
Here, we focus on Bayesian networks and on their construction. Some basics on graphical models and Bayesian networks are provided in this Chapter, while the graphical modelling is dealt in Chapter 3. This Chapter is organized as follow: Section 2.1 introduces some elements of graph theory; Section 2.2 introduces concepts of conditional independence and independence graphs; finally, Section 2.3 deals with Bayesian networks and their properties.

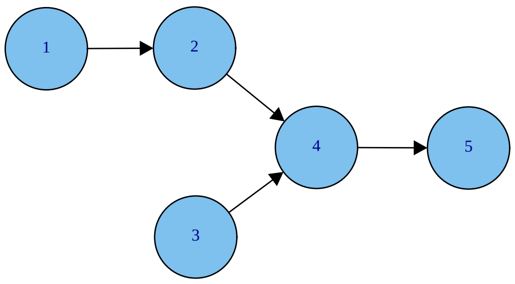
2.1 Elements of graph theory

In this section some basics on graph theory are discussed. The first concept introduced is the notion of *graph* (Berge 1973); then other concepts and the nomenclature of graph theory are introduced.
A graph is a pair of sets V and E, denoted by $G = (V; E)$. In detail, V is a finite set of *vertices* or *nodes* and E is the set of *edges* that is a subset of the $V \times V$ set of ordered pairs of distinct nodes.
Given two distinct nodes a and $b \in V$, the edge $(a; b) \in E$ is called *undirected* if both $(a; b)$ and $(b; a) \in E$ or *directed* if $(a; b) \in E$ but $(b; a) \notin E$. Undirected edges are displayed in the graph by lines, for instance $a - b$, while directed edges by arrows, for instance $a \rightarrow b$. If the graph has only undirected edges, the graph is an *undirected* graph; if it has only directed edges is a *directed* graph. An example of undirected and directed graph is displayed in Figure 2.1. Graphs that have both directed and undirected edges are called *hybrid* graphs.

Let a and b be two generic nodes in a graph. If there is a line between them, $a - b$, they are said to be *adjacent* or *neighbours* and they are denoted by $a \sim b$; if a is linked to b by an arrow, a is said to be *parent* of b, while b is said to be *child* of a; if there is a neither line or arrow between a and b, they are *non-adjacent*, i.e. $a \nsim b$. For instance nodes 1 and 2 in Figure 2.1(a) are adjacent since they are connected to each other by a line; node 1 in Figure 2.1(b) is parent of node 2; it follows that the node 2 is the child of the node



(a) An undirected graph



(b) A directed graph

Figure 2.1: An undirected graph and a directed graph

1. In both Figures 2.1(b) and 2.1(a) nodes 1 and 3 are non-adjacent since no edge exists between them. A graph is *complete* if there is an edge between each pair of nodes.
Let A be a subset of V, $A \subseteq V$, the set of parents, children or neighbours of nodes in A are respectively denoted by:

- $pa(A) = \bigcup_{a \in A} pa(a) \setminus (A)$;
- $ch(A) = \bigcup_{a \in A} ch(a) \setminus (A)$;
- $ne(A) = \bigcup_{a \in A} ne(a) \setminus (A)$.

In addition it is possible defining the *boundary* of A, $bd(A)$, as the set of nodes in $V \setminus A$ that are parents, or neighbours of nodes in A and the *closure* of A, $cl(A)$, as the set constituted by A and its boundary:

- $bd(A) = pa(A) \cup ne(A)$;
- $cl(A) = A \cup pa(A)$;

For instance, let A be the subset of nodes $A : \{4, 5\}$ in the graph of Figure 2.1(a). The set of neighbours and the boundary of A are the following:

- $ne(A) = bd(A) = \{2, 3\}$.
- $cl(A) = \{2, 3, 4, 5\}$;

For the subset $A : \{2, 4\}$ in the graph of Figure 2.1(b), the set of parents, children and the closure of A are the following:

- $pa(A) = bd(A) = \{1, 3\}$.
- $ch(A) = 5$;
- $cl(A) = \{1, 2, 3, 4\}$;

Any subset $A \subseteq V$ induces a subgraph $G_A = (A; E_A)$ where the set E_A consists in those edges of E that have both endpoints in A. If the induced graph is complete, then the subset $A \subseteq V$ is called *complete subset*. A *clique* is a subset $A \subseteq V$ maximally complete, i.e. it becomes incomplete adding another node.

A sequence of n distinct nodes $a = v_o, v_1, ..., v_n = b$ such that $(v_{i-1}, v_i) \in E$ for all $i = 1, ..., n$ is called *path* of length n between a and b. The path that leads from a to b is denoted by $a \mapsto b$ and it is said to be *descendant* if edges are all oriented in the same direction (or some of them are oriented in the same direction and others are undirected). If there is a descendant path between a and b, a is called *ancestor* of b and b *descendant* of a. For instance, consider the graph in Figure 2.1(b): the path $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$ is a descendant path since edges are all oriented in the same direction; the path $1 \rightarrow 2 \rightarrow 4 \leftarrow 3$ is not descendant since there is an inversion of direction in the path. In a descendant path, $a \mapsto b$, the set of all ancestors of b is denoted by $an(b)$ and the set of all descendants of a is denoted by $de(a)$. Two nodes, a and b are connected to each other if there is a descendant path from a to b and a descendant path from b to a. The set of *non-descendants* is denoted by $nd(a) = V \setminus (de(a) \cup a)$.

A $n - cycle$ is a path of length n where $a = b$, i.e. the starting point and the

ending point coincide. The cycle is directed if it contains all arrows equally oriented.
Let A be a subset of V, if $bd(a) \subseteq A \ \forall a \in A$, then A is an *ancestral* set. The smallest ancestral set containing A is denoted by $An(A)$. The notion of ancestral set is relevant for reading conditional independences in directed graphs.
Before concluding this introduction about theoretical aspects of graphs we define an additional algebraic element providing a matrix representation of graphs: *adjacency matrix*. The matrix denotes which vertices are linked to each other by an edge in a given graph. It can be defined for every kind of graph. Here, we only focus on matrices representing some typologies of graph that we are going to handle. In this dissertation, we deal with:

- *undirected graphs*. An undirected graph, from now on UG, has all undirected edges and it is generically denoted by $G = (V; E)$;
- *directed acyclic graphs*. A directed acyclic graph, from now on DAG, has all directed edges and no directed cycle, i.e. it is not possible starting from a node, go back to the same node following arrow directions. Usually, a DAG is denoted by $G^D = (V; E^D)$ where D stands for *directed*.

We firstly define the adjacency matrix for undirected graphs.

Definition 2.1 (Adjacency matrix)**.** Let $G = (V; E)$ be an undirected graph on K nodes. The *adjacency matrix* of G is the $K \times K$ matrix, denoted by $\mathcal{A}$, whose entries $\mathcal{A}_{ij}$ are given by

$$\mathcal{A}_{ij} = \begin{cases} 1 & \text{if and only if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Some properties of this matrix are:

- $\mathcal{A}_{ii} = 0$ for any node i in V;
- if $\mathcal{A}_{ij} = \mathcal{A}_{ji} = 0$ then $i \not\sim j$ in G and these zeros are also called *structural zeros*;

- if $\mathcal{A}_{ij} = \mathcal{A}_{ji} = 1$ then $i \sim j$ in G;
- the matrix $\mathcal{A}$ is real and symmetric.

Consider the graph in Figure 2.1(a); the matricial representation of this graph is given by the following adjacency matrix:

$$\mathcal{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

For DAGs, adjacency matrix can be defined as well. In the directed case, entries of $\mathcal{A}$ are given by:

$$\mathcal{A}_{ij} = \begin{cases} 1 & \text{if and only if } (i \to j) \in E \\ 0 & \text{otherwise} \end{cases}$$

In this case, some of the previous properties do not hold. In particular, the matrix is still real but, clearly, not symmetric. For instance, the adjacency matrix associated with DAG in Figure 2.1(b) is the following:

$$\mathcal{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

An equivalent representation of graph based on a matrix has been provided by Cox and Wermuth (2004). They define the *edge matrix*, here denoted by $\mathcal{E}$, as a triangular binary matrix whose entries $\mathcal{E}_{ij}$ are 1 if and only if $j \to i$ or $j = i$ in the represented graph. As a consequence of this, the edge matrix is a transpose of adjacency matrix with additional ones along the diagonal, i.e.:

$$\mathcal{E} = \mathcal{A}^T + I$$

where the notation $\mathcal{A}^T$ denotes the transpose matrix of $\mathcal{A}$ and I the identity matrix.
This Section concludes providing a last notion for DAGs. Let $G^D = (V; E^D)$ be a DAG; it is often useful referring to the *moralized* version of the graph. The moral graph associated to G^D is a graph $G^m = (V; E^m)$ having the same nodes of G^D and a new set of edges, denoted by E^m. The set E^m consists of edges in E, taken without considering directions, and edges obtained by adding an undirected link between nodes with a common child. The moral graph plays a crucial role for interpreting conditional independences in directed graphs (see Section 2.2.3). In the following Section, we introduce conditional independence notion and some properties.

2.2 Conditional independence

As previously said, a graphical model provides a pictorial representation of independence structure of a set of variable, so it combines a probabilistic and a graphical aspect. In the previous Section, some basics on graph theory have been introduced; this Section deals with the *conditional independence* concept that is the probabilistic aspect of graphical models.
Conditional independence has been formally discussed by Dawid (1980). We provide the following definition.

Definition 2.2 (Conditional independence)**.** Let X, Y and Z three random variables with joint distribution P; X is conditionally independent of Y given Z (with respect to P), if for any measurable set A in the sample space of X, there is a version of the conditional probability $P(A|Y, X)$ which is a function of Z alone.

According to the notation due to Dawid (1979), we write $X \perp\!\!\!\perp Y|Z[P]$ to say that X is conditionally independent of Y given Z under P. Generally, P is fixed and we can use the simplified notation $X \perp\!\!\!\perp Y|Z$.
For three discrete random variables, X, Y and Z, the condition $X \perp\!\!\!\perp Y|Z$

simplifies as

$$P\{X = x; Y = y|Z = z\} = P\{X = x|Z = z\}P\{Y = y|Z = z\} \qquad (2.1)$$

whenever $P\{Z = z\} > 0$
The factorization (2.1) is equivalent to several characterizations that are:

1. $X \perp\!\!\!\perp Y|Z \iff P\{X = x; Y = y; Z = z\} = \frac{P\{X=x;Z=z\}P\{Y=y;Z=z\}}{P\{Z=z\}}$
2. $X \perp\!\!\!\perp Y|Z \iff P\{X = x|Y = y; Z = z\} = P\{X = x|Z = z\}$
3. $X \perp\!\!\!\perp Y|Z \iff P\{X = x; Z = z|Y = y\} = P\{X = x|Z = z\}P\{Z = z|Y = y\}$
4. $X \perp\!\!\!\perp Y|Z \iff P\{X = x; Y = y; Z = z\} = h(X = x; Z = z)k(Y = y; Z = z)$
5. $X \perp\!\!\!\perp Y|Z \iff P\{X = x; Y = y; Z = z\} = P\{X = x|Z = z\}P\{Y = y; Z = z\}$

Different expressions lead to some considerations about conditional independence largely discussed in the literature (Dawid 1979; Pearl 1988). We only focus on some of them. The first characterization states that the joint distribution can be expressed as the product of marginal distributions of conditional independent variables. The fourth expression states that the factorization can involve also two generic functions non necessary coincident with marginals. Another relevant aspect is that associated with the second formulation: if $X \perp\!\!\!\perp Y|Z$, the distribution of X given Y and Z is entirely determined by Z alone. This implies that Y is superfluous for determining X once Z is given. This assertion recalls the logical concept of *irrelevant information*. In detail, the conditional independence relation satisfies several properties all interpretable in term of relevance that are:

(C1) if $X \perp\!\!\!\perp Y|Z$, then $Y \perp\!\!\!\perp X|Z$;

(C2) if $X \perp\!\!\!\perp (Y \cup W)|Z$, then $X \perp\!\!\!\perp Y|Z$ and $X \perp\!\!\!\perp W|Z$;

(C3) if $X \perp\!\!\!\perp (Y \cup W)|Z$, then $X \perp\!\!\!\perp Y|(Z \cup W)$;

(C4) if $X \perp\!\!\!\perp Y|Z$ and $X \perp\!\!\!\perp W|(Y \cup Z)$, then $X \perp\!\!\!\perp (W \cup Y)|Z$.

These properties, formally discussed by Pearl (1988) and by Geiger and Pearl (1993), can be interpreted as logical axioms of conditional independence. In detail, $(C1)$ property captures the *symmetry* property: given Z, if X does not give any additional information about Y, then Y is irrelevant for knowing X. $(C2)$ stands for the *decomposition* axiom asserting that given Z, if two combined sets, Y and W, are judged irrelevant to X, then each of them is still irrelevant to X once taken separately. The *weak union* property is encoded in $(C3)$: given Z, learning the irrelevant information W does not allow to the irrelevant information Y to become relevant to X. Finally, $(C4)$ property deals with the *contraction* axiom: if W is non informative to X after having learnt Y and Z, then W must have been irrelevant to X also before having learnt Y. The last two properties suggest that irrelevant information does not alter the relevance state of other information.
Under the condition of positiveness of P, conditional independence satisfies a further property that is the *intersection* axiom.

(C5) if $X \perp\!\!\!\perp Y|(Z \cup W)$ and $X \perp\!\!\!\perp W|(Y \cup Z)$, then $X \perp\!\!\!\perp (Y \cup W)|Z$.

This property states that, if X and Y are separated by a set $S_1 = Z \cup W$ and if X and W are separated by a set $S_2 = Y \cup Z$, then the only intersection of S_1 and S_2, i.e. Z, separates X and $(Y \cup W)$.

$(C1) - (C4)$ properties are also called *semi-graphoid* axioms and, as a consequence of this, an algebraic structure that satisfies these axioms is thus said to be a *semi-graphoid*; $(C1) - (C5)$ are called *graphoid* axioms and an algebraic structure that satisfies them is called *graphoid*. The main advantage of semi-graphoids and graphoids concerns their graphical representation that makes easy reasoning about conditional independence. In order to graphically interpret conditional independence, it is necessary to introduce further elements that are discussed in the following Section.

2.2.1 Independence graphs and Markov properties

Graphical models are an efficient and intuitive language that displays conditional independence by graphs. A graph $G = (V; E)$ is thus a representation of a model or, more specifically, of an *independence model*. An independence model, generally denoted by $\mathcal{M}$, is a set of conditional independence relations and it is a graphoid whenever it satisfies the $(C1) - (C5)$ axioms.
Graphically, the conditional independence statements can be portrayed by *separation* concept in graphs.

Definition 2.3 (Separation)**.** Let A, B and S be three disjoint subsets of nodes in a graph $G = (V, E)$. If every path between a node in A and a node in B passes through at least one node in S, then A is separated from B by S.

For instance, in Figure 2.1(a) nodes in the set $A = \{1, 2\}$ are separated from $B = \{5\}$ by the set $S = \{3, 4\}$. Furthermore $A \perp\!\!\!\perp B|S$ is a graphoid since it is an algebraic structure that satisfies the graphoid axioms.
The separation property is similar to the decomposition and the weak union. However, the separation is stronger then $(C2)$ and $(C3)$ properties. The decomposition states that a single variable X may be independent of a variable in the set Y even if it is dependent of the whole set; the separation reflects that two sets of vertices are separated if there is no path between an element of one set and an element of the other set. Furthermore, given a graph, if $A \perp\!\!\!\perp B|S$, the separating set S can be enlarged without modifying the independence relation between A and B; the $(C3)$ property, instead, sets out the conditions under which the conditional independence holds.
However, in some situation, vertex separation does not satisfy all conditional independence dependence relations embodied in the model. Because of this weakness, it is necessary to portray either conditional independences or conditional dependences in a single graph that becomes a map of the model. Generally speaking, an undirected graph G is a dependence map, *D-map*, of $\mathcal{M}$ if there is a one-to-one correspondence between the elements of the model and the nodes V of the graph. This correspondence ensures that the graph is able to display some model properties. Formally speaking, G is a *D-map*

if for any triplet of disjoint subsets of variables, A, B and S, verifying the conditional independence relation between A and B given S in the model $\mathcal{M}$, then the same relation holds in the graph G as well. Denoting by:
$A \perp\!\!\!\perp_{\mathcal{M}} B|S \Leftrightarrow$ A and B are conditionally independent given S in the model $\mathcal{M}$,
$A \perp\!\!\!\perp_{G} B|S \Leftrightarrow$ A and B are conditionally independent given S in the graph G,
we can write that G is a D-map of $\mathcal{M}$ if:

$$A \perp\!\!\!\perp_{\mathcal{M}} B|S \Rightarrow A \perp\!\!\!\perp_{G} B|S.$$

On the contrary, the model can be approximated by an *independence map*, I-map. If A is separated from B by S in an I-map, then A is conditionally independent by B given S in the model $\mathcal{M}$:

$$A \perp\!\!\!\perp_{G} B|S \Rightarrow A \perp\!\!\!\perp_{\mathcal{M}} B|S.$$

In I-maps each independence relation modelled in the graph has to be consistent with the joint probability distribution P. However, the probability distribution might imply additional independences not represented in an I-map. A more interesting structure is thus a *minimal* I-map that does not encode spurious relations. An I-map of a graphoid is *minimal* if it immediately stops to be an I-map deleting an edge. A minimal I-map obtained starting from a complete graph and by deleting every edge (a, b) for which X_a and X_b are conditionally independent in the model is also called *conditional independence graphs* (Pearl 1988).

Definition 2.4 (Conditional independence graph)**.** Consider a set of random variables associated with nodes V in a graph; the undirected graph $G = (V; E)$ is a conditional independence graph if for every missing edge in G a conditional independence statement holds, i.e.

$$(a, b) \notin E \Longleftrightarrow X_a \perp\!\!\!\perp X_b|X_{V\setminus\{a,b\}}$$

It is worth noting that a graph G is said to be a *perfect map*, P-map for short, if it is both a D-map and a I-map, i.e.

$$A \perp\!\!\!\perp_{\mathcal{M}} B|S \Longleftrightarrow A \perp\!\!\!\perp_G B|S.$$

In this case, the model $\mathcal{M}$ is said to be *graph-isomorph* since there exists a graph G that is a perfect map of $\mathcal{M}$.
The formal link between conditional independence in probability and the separation in graphs is given by *Markov* properties. We will discuss Markov properties firstly with respect to undirected graphs and then with respect to directed acyclic graphs.

2.2.2 Markov properties on undirected graphs

Different aspects of conditional independence can be caught by Markov properties that merge conditional independence in probability and separation in graphs.
Let $G = (V, E)$ be a graph and denote by $(X_a)_{a \in V}$ the set of random variables associated with V taking values in probability space $(\mathcal{X}_a)_{a \in V}$. Note that to simplify the notation we will denote the generic vector X_A, with its index A. So, we will write $A \perp\!\!\!\perp B|C$ instead of $X_A \perp\!\!\!\perp X_B|X_C$.
The joint probability distribution P satisfies the following Markov properties according to G:

(P) *Pairwise Markov property*: for any pair (a, b) of non-adjacent nodes

$$a \perp\!\!\!\perp b|V \setminus \{a, b\}$$

(L) *Local Markov property*: for any vertex $a \in V$

$$a \perp\!\!\!\perp V \setminus \{cl(a)\}|bd(a)$$

(G) *Global Markov property*: for any triplet (A, B, S) of disjoint subsets $\in V$

$$A \perp\!\!\!\perp B|S$$

The Pairwise property states that two non adjacent variables are conditionally independent to each other given the rest. This property represents the starting point for building the independence graph since this has been defined with respect to a pair of nodes. The Local Markov properties, instead, is closely related to prediction and has the following meaning: a variable can be explained by its neighbour variables corresponding to its boundary. Finally, the Global Markov property provides a general criterion for simple translating from the separation to the conditional independence. It is worth noting that the *(G)* property is the strongest one. This determines that the list of conditional independences implied by *(G)* contains the statements associated with *(P)* and *(L)*.
For any undirected graph G and any probability distribution P on $\mathcal{X}$ it holds that:

$$(G) \Longrightarrow (L) \Longrightarrow (P).$$

The contrary does not hold in general; however, Pearl and Paz (1987) demonstrated that, if $(C5)$ property is verified, then (P) implies (L) that implies (G) and the Markov properties are equivalent:

$$(G) \Longleftrightarrow (L) \Longleftrightarrow (P).$$

Both Markov properties and conditional independence are related to factorization criterion that can be formally defined as follows.

Definition 2.5 (Factorization)**.** A probability measure P on $\mathcal{X}$ is said to *factorize* according to G if for all complete subsets $A \in V$ there exists a non-negative function ψ_A depending on elements of $\mathcal{X}_A$ only, and there exists a product measure μ so that

$$f(x) = \prod_A \psi_A(x).$$

Without loss of generality, it is possible to express the factorization in terms of cliques since each clique c in the set of cliques C is a maximally complete subset:

$$f(x) = \prod_{c \in C} \psi_A(x).$$

If P factorizes, we say that P has the property (F). (F) implies (G), but when P verifies the $(C5)$ property, all Markov properties are equivalent (Speed 1979):

$$(F) \iff (G) \iff (L) \iff (P).$$

2.2.3 Markov properties on directed acyclic graphs

In presence of DAGs, Markov properties assume a different formulation that has been systematically studied by Kiiveri *et al.* (1984) and by other several authors (Pearl and Verma 1987; Geiger and Pearl 1990; Lauritzen *et al.* 1990; Verma and Pearl 1990a).
Firstly, the separation criterion needs to be redefined according to DAGs. More specifically, it admits two equivalent formulations: the *d-separation* criterion introduced by Pearl (1986) and the criterion based on the *moral graph* due to Lauritzen *et al.* (1990). We start dealing with the d-separation criterion.

Definition 2.6 (d-separation)**.** Let A, B and S be three disjoint subsets of a DAG G^D. S d-separates A from B, if every path between a node in A and a node in B is *blocked* by a node in S and we write $A \perp\!\!\!\perp_D B|S$.

The key concept of this definition concerns the notion of *blocked* path. Let $G^D = (V; E^D)$ be a DAG and let π be a path between two nodes in the graph, say $a \in A$ and $b \in B$; π is said to be blocked by S if one of the following two conditions is satisfied:

1. there is a node c in π having serial ($\rightarrow c \rightarrow$) or diverging ($\leftarrow c \rightarrow$) connection in S, or;

2. any node c in π having converging connection ($\rightarrow c \leftarrow$) neither is not in S nor has descendants in S.

An equivalent formulation of the d-separation has been provided by Cox and Wermuth (1996). They use the alternative concept of *active* path. A path in a DAG is active if it is not blocked, thus if:

1. every node with converging connection either is or has a descendant in S, or;

2. every other node is outside S.

The d-separation criterion is a tool for finding conditional independences in a DAG. The method can be really complex to apply when the graph is large. A more suitable technique is based on the moral graph associated with the DAG induced by the set $An(A, B, S)$. The moral graph has been defined in Section 2.1 as the undirected version of a DAG obtained marrying parents with common children and ignoring arrow directions of every edge. Following this method, firstly it is necessary to build the moral graph of the smallest ancestral set of (A, B, S); once having transformed the DAG into its moral graph, the conditional independences are read off in the moral graph as in the undirected case.
It is worth noting that both criteria lead to the same conclusion as it is shown in the following example.
Consider the DAG in Figure 2.2. Suppose we are interested in verifying if $1 \perp\!\!\!\perp 6|3$ and $1 \perp\!\!\!\perp 6|(3, 4, 5)$.

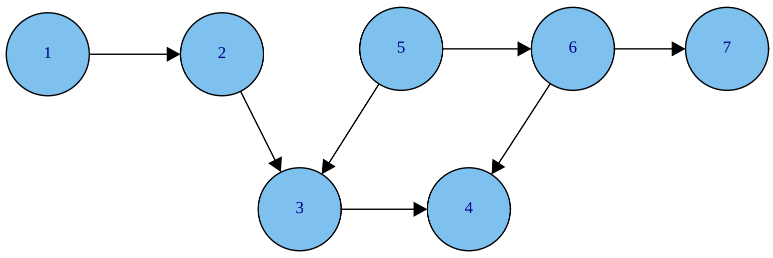
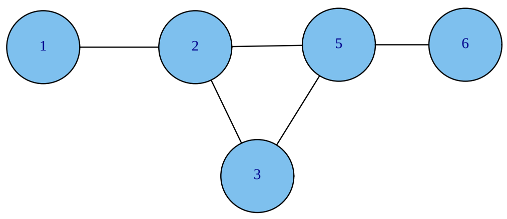


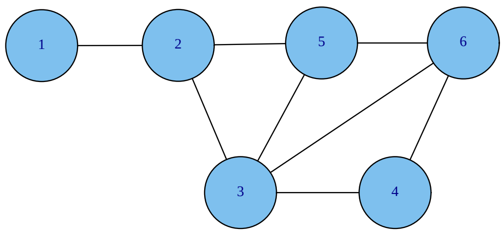
Figure 2.2: A DAG on 7 nodes

Moral graphs associated with $An(1, 3, 6)$ and $An(1, 3, 4, 5, 6)$ are displayed in Figure 2.3.

Observing those graphs it is possible to state that $1 \not\!\perp\!\!\!\perp 6|3$ (note the alternative path $1 - 2 - 5 - 6$ in Figure 2.3(a)) and that $1 \perp\!\!\!\perp 6|(3, 4, 5)$ (see



(a) $(G_{An(1,3,6)})^m$



(b) $(G_{An(1,3,4,5,6)})^m$

Figure 2.3: Moral graphs

Figure 2.3(b)).
The d-separation criterion leads to the same conclusion: 1 is not d-separated from 6 by 3 since the node 3 in the path $1 \rightarrow 2 \rightarrow 3 \leftarrow 5 \rightarrow 6$ belongs to the separator set, thus the path is active; 1 is instead d-separated from 6 by (3,4,5) since all paths between 1 and 6 are blocked by the conditioning set.

Having discussed the concept of separation for directed acyclic graphs, it is possible introducing the Markov properties for DAGs.
Let G^D be a DAG, the probability measure P obeys to the following properties:

(DP) *Directed Pairwise Markov property*: for any pair (a, b) of non-adjacent nodes with $b \in nd(a)$

$$a \perp\!\!\!\perp b | nd(a) \setminus \{b\}$$

(DL) *Directed Local Markov property*: for any vertex $a \in V$

$$a \perp\!\!\!\perp \{nd(a) \setminus pa(a)\} | pa(a)$$

(DG) *Directed Global Markov property*: for any triple (A, B, S) of disjoint subsets $\in V$

$$A \perp\!\!\!\perp B|S \text{ in } G^m_{An(A,B,S)}$$

If the graph were undirected, $de(a) = 0$ so that $nd(a) = V \setminus \{a\}$ and $pa(a) = ne(a)$. As a consequence of this, (DP) would be equal to (P) and (DL) to (L). Lauritzen *et al.* (1990) demonstrate that, in the directed case, (DL) and (DG) are equivalent, even without any assumption on probability distribution. This allows to build an I-map starting from local dependences directly and bypassing the (DP) property that is the weakest Markov property in the directed case. It is worth noting that a DAG is an I-map of probability distribution P for a set of variables if every d-separation displayed in the graph entails a conditional independence in the distribution.
For DAGs, the probability distribution P factorizes according to G^D if the joint probability can be expressed as the product of the conditional distribution of each node given its parents:

$$p(x) = \prod_{i=1}^{K} p(x_i|pa(x_i)). \tag{2.2}$$

In this case, we say that P admits a *recursive factorization* and has (DF) property. Expression (2.2) represents the reduced form of another formulation that can be discussed after having introduced the *well-ordering* of vertices.
Consider a numbering of the vertices V in G^D so that $(a, b) \in E^D$ implies that $number(a) < number(b)$, i.e. b is parent of a. Such an ordering is called *well-ordering* and it allows to define, for a generic node b, the set of *predecessors*. This is usually denoted by $pr(b)$ and it is the set of nodes that have a numbering less to the numbering of b. Considering a given well-ordering, the probability distribution P can be expressed as follow:

$$p(x) = p(x_K|x_{K-i}, x_{K-2}, ..., x_1) \cdot ... \cdot p(x_2|x_1) \cdot p(x_1),\ n = |V| \tag{2.3}$$

where $p(x_i|x_{i-1}, x_{i-2}, ..., x_1) = p(x_i|pa(x_i))$. The expression (2.3) is called *chain rule* and it reduces to (2.2). Furthermore, it implies that P satisfies the relation $X_a \perp\!\!\!\perp X_{pr(a)}|pa(a)$, i.e. the *Directed Ordered Markov property*, (DO) for brevity.

(DO) *Directed Ordered Markov property*: for any node a in V

$$a \perp\!\!\!\perp pr(a)|pa(a)$$

The well-ordering in a DAG is ensured by its acyclicity.
In the directed case, Markov properties are all equivalent, just assuming the existence of a density (Cowell *et al.* 1999)

$$(DF) \iff (DG) \iff (DL) \iff (DO).$$

2.3 Bayesian networks and further properties of DAGs

In previous Sections some elements of graph theory and some probabilistic aspects of graphical models have been discussed. This Section deals with some graphical models based on DAGs and called Bayesian networks; their remarkable properties and some further aspects of DAGs are discussed.
Generally speaking, a Bayesian network is a probabilistic model for representing the multivariate probability distribution of a set of variables. The pictorial representation of the model is provided by a DAG where nodes represent variables of the model and arrows stand for influence directions between variables.

Definition 2.7 (Bayesian network)**.** Let P be a multivariate probability distribution of a set random variables $\boldsymbol{X}$ and let $G^D = (V; E^D)$ be a DAG. The pair (G^D, P) is a Bayesian network if it satisfies the Directed Local Markov property.

We could also refer to the (DL) property with the name of *Markov condition* (Pearl 2000). As discussed in the previous Section, it represents the starting point for building an I-map of a probability distribution since it entails that the missing edge between two nodes means there is not a direct dependency between their associated variables. However, it would be interested also saying that the presence of an edge between two nodes means a direct dependency between variables. This is entailed by the *faithfulness* condition (see Neapolitan (2003) for a formal and detailed treatment of the condition).

Definition 2.8 (Faithfulness)**.** Let P be a joint probability distribution of a set of random variables and let $G^D = (V, E^D)$ be a DAG. The pair (G^D, P) satisfies the faithfulness condition if, based on the Markov condition, G^D entails all and only conditional independences in P.

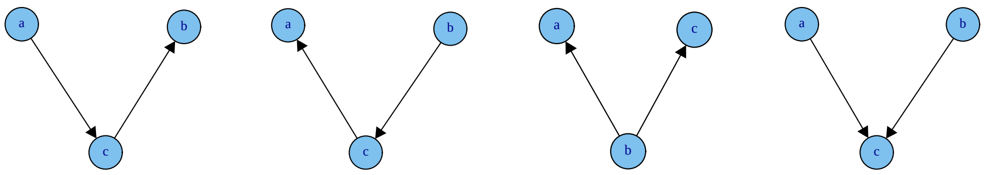
This condition states that:

- G^D entails only conditional independences in P (so the Markov condition is verified),
- all conditional independences in P are entailed by G^D.

When the faithfulness condition is satisfied, P and G^D are said to be faithful to each other. In other words, the faithfulness condition is verified if and only if G^D is a perfect map of P. As a consequence of this, if a probability distribution is faithful then the perfect map is uniquely determined. The faithfulness condition is crucial when the DAG structure is learnt directly from data. In fact, the correctness of several algorithms for learning the DAG structure has been proved under the faithfulness condition.
As just said, if the faithfulness condition is verified, then P satisfies this condition with G^D. Furthermore, P satisfies the condition also with those DAGs that are *Markov equivalent* to G^D or, generically speaking, that encode the same d-separations of G^D even though they have a different graphical structure. In fact, the conditional independences encoded in a DAG, identifiable by the d-separation criterion, implies a unique set of probability distributions, but the set of probability distributions does not determine a unique DAG. Hence, different DAGs can encode the same conditional

independence relations.
For instance, consider DAGs in Figure 2.4. Different situations involving a pair of nodes, a and b, directly connected with the node c are displayed. Node c plays the rule of *transition* node in the serial configurations, *common source* node in the diverging structure or *common sink* in the converging structure. A common sink is also called *collider*.

Definition 2.9 (Collider)**.** A node c is a collider in a path if c has two incoming edges.



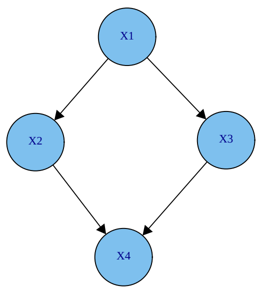
(a) serial configuration (b) serial configuration (c) diverging configuration (d) converging configuration

Figure 2.4: Different configurations for a triplet of nodes so that the first three DAGs belong to the same equivalence class and the fourth DAG is contained in a different one

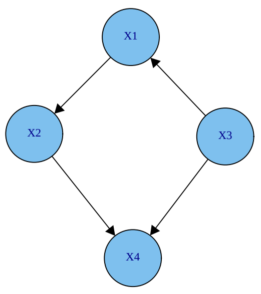
Serial and diverging configurations encode both the relations $a \not\perp\!\!\!\perp b$ and $a \perp\!\!\!\perp b|c$ so that DAGs (a), (b) and (c) encode the same d-separations. On the contrary, DAG (d) shows a converging connection, also called head-to-head configuration or *v-structure* following the notation of Cox and Wermuth (1996), and it entails a different relation that is $a \not\perp\!\!\!\perp b|c$. It follows that DAGs with the same d-separation properties are said *Markov equivalent*. The definition of Markov equivalence for DAGs is due to Verma and Pearl (1990b).

Definition 2.10 (Markov equivalence)**.** Two DAGs are equivalent if and only if they have the same skeleton (that is the structure of the graph dropping directions) and the same v-structures.

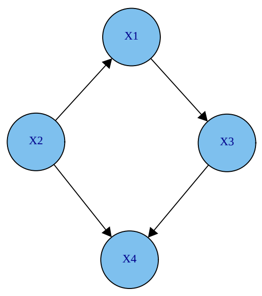
For instance, consider DAGs in Figure 2.5. They have the same set of nodes V representing a set of variables $\boldsymbol{X} = (X_1, X_2, X_3, X_4)$ but a different set of edges. Nevertheless, they are equivalent since they have the same basic structure, displayed in Figure 2.6 and the same converging connection in node X_4.



(a) A DAG with $E = X_1 \to X_2, X_1 \to X_3, X_2 \to X_4, X_3 \to X_4$



(b) A DAG with $E = X_1 \to X_2, X_3 \to X_1, X_2 \to X_4, X_3 \to X_4$



(c) A DAG with $E = X_2 \to X_1, X_1 \to X_3, X_2 \to X_4, X_3 \to X_4$

Figure 2.5: Markov equivalent graphs

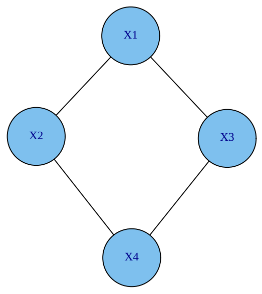


Figure 2.6: Skeleton of DAGs in Figure 2.5

The Markov equivalence permits to partition the space of DAGs into classes of models, namely *equivalence classes*, containing all the DAGs which are Markov equivalent to each other. The notion of equivalence is of particular interest when the Bayesian network is learnt from data. Bayesian networks belonging to the same equivalence class are statistically indistinguishable since they represent equivalent parameterizations of the same distribution (Chickering 1995). As a matter of fact, the probability distribution implied by data can determine only an equivalence class rather than a specific

DAG[1]. Thus, it is useful to provide a representation of equivalence class with a single graph. More specifically, according to the definition of Markov equivalence, we can display an equivalence class by an hybrid graph having the same skeleton and the same v-structures as the DAGs in the class. Such a graph is called *Partially* DAG (PDAG)[2] A PDAG, usually denoted by *gp*, contains both directed edges (between nodes involved in v-structures) and undirected edges (between other nodes). For instance, the PDAG, representing the Markov equivalence class to which DAGs in Figure 2.5 belong, is displayed in Figure 2.7.

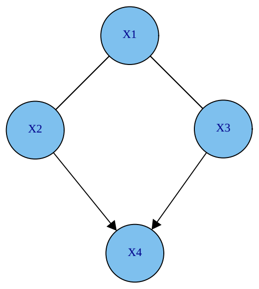


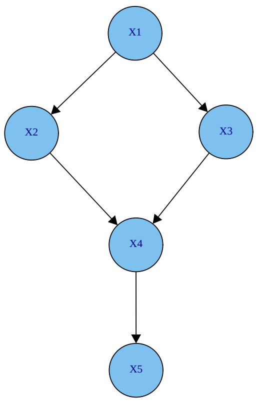
Figure 2.7: PDAG of DAGs in Figure 2.5

It is worth noting that there are often additional edges, other than those involved in head-to-head configurations, that has to be oriented in the PDAG. For example, if all DAGs in the equivalence class have the edge $X_a \rightarrow X_c$ and the same configuration $X_a \rightarrow X_c - X_b$ that is not a v-structure, then all the DAGs in the equivalence class must have the edge $X_c - X_b$ oriented as $X_c \rightarrow X_b$. In detail, only edges with the same directions in all networks of the class are directed in PDAG, while edges differently orientated in networks are displayed as undirected. Edges with invariant orientations in all structures are said to be *compelled* and other edges are said to be *reversible*. Given a generic *gp*, the set of compelled edge in graph is usually denoted by C_G while the set of reversible edges by R_G. Consider DAGs in Figure 2.8. They are Markov equivalent to each other; none of them could have

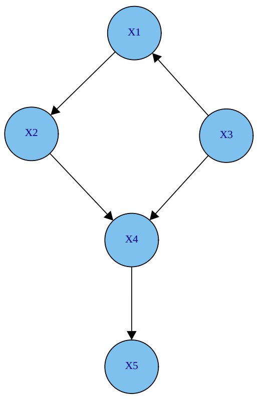
[1]Notice that equivalence classes can be used instead of DAGs for reducing the searching space in the structural learning phase. More details are discussed in the following Chapter dealing with network structures learning from data.

[2]PDAG is also called DAG pattern (Neapolitan 2003).

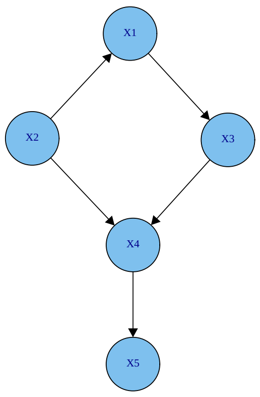
the undirected edges between nodes X_4 and X_5 because this would create another head-to-head configuration. So, the set of compelled edges in PDAG is $C_G = (X_2 \to X_4, X_3 \to X_4, X_4 \to X_5)$.



(a) A DAG with $E = X_1 \to X_2, X_1 \to X_3, X_2 \to X_4, X_3 \to X_4, X_4 \to X_5$



(b) A DAG with $E = X_1 \to X_2, X_3 \to X_1, X_2 \to X_4, X_3 \to X_4, X_4 \to X_5$



(c) A DAG with $E = X_2 \to X_1, X_1 \to X_3, X_2 \to X_4, X_3 \to X_4, X_4 \to X_5$

Figure 2.8: Markov equivalent graphs

The PDAG representing the Markov equivalence class to which graphs in Figure 2.8 belong is displayed in Figure 2.9.

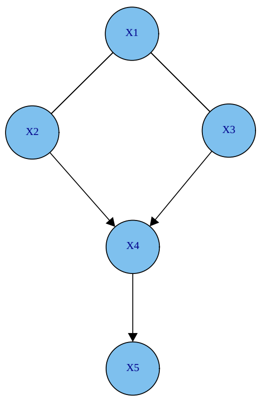


Figure 2.9: PDAG of DAGs in Figure 2.8

We conclude the Chapter providing an additional notion.

A DAG G^D is called a *consistent extension* of PDAG *gp* if:

- G^D has the same skeleton and the same v-structures of PDAG;
- all edges directed in PDAG are directed in G^D;
- G^D does not contain any uncoupled head-to-head meetings that are not in PDAG.

A PDAG admits a consistent extension if there is at least a DAG that is a consistent extension of the PDAG. So, any DAGs in the same Markov equivalence class is a consistent extension of the PDAG representing that equivalence class.

Chapter 3

Structural learning

Building a Bayesian network requires to define three components (Cowell *et al.* 1999):

- *qualitative* part that stands for the graphical structure of the model. The directed graph represents variables of interests by nodes and displays relationships between them by arrows. In detail direct edges represent direct relevance of one variable to another;

- *probabilistic* stage that requires to specify, according to the DAG, the joint distribution defined on the variables;

- *quantitative* stage that is the numerical specification of probability tables associated with each node in the graph.

These steps can be performed with the help of experts that are able to suggest conditional independence relations between variables according to the domain theory and to the previous knowledges. However, in many situations domain expertise is not available and, more often, the domain graphical structure is unknown (or partially known). In these cases, building the network manually may become a really complex task to perform and the model has to be estimated from data.
Model selection in Bayesian networks consists in determining the DAG structure representing the joint probability distribution implied by the avail-

able data. This phase, namely *structural learning*, has been largely discussed in the literature (Buntine 1994; Buntine 1996; Neapolitan 2003) and it mainly can be supported through two approaches: *scoring and searching* and *constraint-based*. Both approaches have advantages and limitations that are briefly introduced below.

Scoring and searching

Algorithms following the scoring and searching approach (Cooper and Herskovits 1992; Heckerman 1995) span the space of all possible models $\mathcal{M}$ that can be built for a set of random variables $\boldsymbol{X}$. First a score function is chosen and a score is computed for all possible models, then the model having the maximum score is selected.
The main advantage of scoring and searching methods is that they compare several alternative models using different (Bayesian or non-Bayesian) criteria. One of the most used measures is the maximum (log-)likelihood $l(\hat{\theta}_m)$. For a general model m this is:

$$l(\hat{\theta}_m) = \log(L(\hat{\theta}_m)) := \log p(\mathcal{D}|m, \hat{\theta}_m) \tag{3.1}$$

where θ_m are Bayesian network parameters i.e. the conditional probabilities of the variables given their parents in the DAG. So, this criterion consists in choosing the model that maximises the probability distribution of data $\mathcal{D}$ given the model m and the conditional probabilities of the variables. The (3.1) measure, however, does not consider the complexity of the model. In the estimation phase a trade-off between the complexity of the model and the goodness of fit is often required. Some measures that consider the models complexity are *Akaile Information Criterion (AIC)*(Akaike 1974) and *Bayesian Information Criterion (BIC)*(Schwarz 1978). Both criteria penalize the maximum likelihood value with a term that is represented by the *degree of freedom* d_f of the model in the *AIC* and by $\frac{1}{2} \cdot d_f \cdot log(n)$ in the *BIC* where n is the sample size. The *BIC* measure is essentially similar to another popular measure of goodness of fit coming from information theory, namely *Minimum Description Length (MDL)* (see Rissanen (1978) for furhter details). Many other measures and combinations of techniques (de Campos *et al.* 2002; Blanco *et al.* 2003) can be used to maximise the evaluation of the learnt structure. Nevertheless, finding the optimum DAG is a NP-hard problem since the research space is large (Chickering *et al.* 1994). So, score-and-search methods can get stuck at a local optimum of the scoring function; furthermore, variations proposed to overcome this limitation can be computationally expensive (Steck 2007).

Constraint-based

Constraint-based algorithms, carry out a sequence of independence tests and, according to their results, draw a DAG encoding the learnt conditional independence relations. These methods try to find the DAG that satisfy all and only the conditional independences that can be estimated from data. An advantage of these algorithms is that they are intuitive being based on independence analysis; in addition, under some assumptions (see 3.1.2), a DAG equivalent to the true one is selected. More specifically, they iteratively check (conditional) independences by performing statistical tests on the data. From the beginning, unchangeable decisions about independences are taken according to each test result and, finally, a unique model is found. So, edge presence or absence in the graph is due to the statistical test results that can present mistakes (Moral 2004); a drawback of constraint-based methods is that an erroneous decision in the procedure can influence the algorithm future behaviour and can determine the selection of a suboptimal model.

This work proceeds along the constraint-based approach. Starting from a well-known constraint-based algorithm, the work proposes some variations that are able to improve the structural learning phase in presence of categorical variables. This Chapter is organized as follow: Section 3.1 deals with one of the most popular constraint-based algorithms, PC algorithm: the procedure is explained and some disadvantages are discussed; Section

3.2 introduces some nonparametric independence tests appropriate for categorical variables necessary for new proposals; in Sections 3.3 and 3.4 two new algorithms for ordinal and mix nominal-ordinal variables are presented. Notice that, to simply notation, from now on we will denote a generic DAG $G^D = (V; E^D)$ with $G = (V; E)$.

3.1 PC algorithm

One of the most widespread and well-known algorithms belonging to the constraint-based approach is the *PC algorithm* (Spirtes *et al.* 2000). It is a stepwise backward algorithm that starts from a complete undirected graph and it tests the conditional independences for each pair of nodes in the graph given a separator subset of increasing cardinality. Step by step, the algorithm computes the conditional cross entropy (see Section 3.1.1) between pair of variables and, according to test result, removes or maintains edges between corresponding nodes in the graph.
The consistency of this algorithm is proved under some assumptions:

- DAG and the joint distribution probability P are faithful to each other (see Section 2.3);
- data are infinite;
- statistical tests have no errors.

If these conditions are verified, the algorithm is able to discover a DAG equivalent to the true one. The first condition guarantees that the estimated DAG is a directed perfect map of a given joint probability distribution. This means that the DAG entails all, and only, conditional independences induced from observed data. The second assumption is quite restrictive and does not necessarily happen. Dataset may be not large enough to ensure the correctness of statistical tests and some errors can occur. Furthermore, the number of statistical mistakes increases when the dataset is small or when the conditioning set S is large. If the third assumption is not verified, it is possible that the algorithm does not produce the true graph. Despite these limitations, PC algorithm is largely used since some good properties (asymptotic consistency and correctness) have been proved (Kalisch and Bühlmann 2007). In the following the general procedure of PC algorithm is introduced.

3.1.1 Entropy, cross entropy and conditional cross entropy

PC algorithm checks conditional independence between pairs of variables using some information-theoretic principles. We firstly introduce the concept of entropy developed by Shannon (1948).
The entropy $H(\cdot)$ of a random variable X with k states and probability mass function $P(x)$ is a measure of how much the probability mass function is scattered over the states.

Definition 3.1 (Entropy)**.** The entropy $H(X)$ of a random variable X with probability mass function $P(x)$ is:

$$H(X) = -\sum_{x} P(x) \cdot logP(x).$$

Thus $H(X)$ is a measure of X uncertainty and it only depends on X probabilities (Cover and Thomas 2006). The maximum entropy is achieved when X takes k distinct values each with probability $1/k$ so that $H(X) = log(k)$. The minimum is achieved when all the probability mass is located on a single state so that $H(X) = 0$. Thus, the entropy, is a non-negative quantity $\in [0, log(k)]$.

A measure of the distance between two distributions, say $P(x)$ and $Q(x)$, is the relative entropy also called *Kullback-Leibler* divergence $D(P||Q)$ (Kullback and Leibler 1951).

Definition 3.2 (Relative entropy)**.** The Kullback-Leibler divergence between two probabilities $P(x)$ and $Q(x)$ is

$$D(P||Q) = \sum_x P(x) \cdot \log \frac{P(x)}{Q(x)}.$$

This measure is not symmetric and it is supposed to be equal to zero if and only if $P = Q$. The Kullback-Leibler divergence is infinite if $P(x) > 0$ and $Q(x) > 0\ \forall x$.
Let X and Y be two random variables with joint distribution $P(x, y)$ and marginal distributions $P(x)$ and $P(y)$ respectively. The Kullback-Leibler divergence can be used to measure the distance between the joint distribution and the product of marginal distributions, thus, providing a natural measure of dependence. The relative entropy between the joint distribution $P(x, y)$ and the product $P(x) \cdot P(y)$ is said cross entropy between X and Y.

Definition 3.3 (Cross entropy)**.** The cross entropy between two random variables X and Y with joint distribution $P(x, y)$ and marginal distributions $P(x)$ and $P(y)$ is:

$$CE(X, Y) = \sum_{x,y} P(x, y) \cdot log \frac{P(x, y)}{P(x)P(y)}. \tag{3.2}$$

Cross entropy takes non-negative values and it can be equivalently expressed as:

$$CE(X, Y) = H(X) - H(X|Y)$$

where $H(X)$ is the entropy of X and $H(X|Y)$ is the entropy of X given Y. It is worth noting that (3.2) can be interpreted as the mutual information between two variables, that is the information quantity gained about one variable from having observed the other. Cross entropy is also symmetric since it quantifies the mutual dependence between two variables, that is the amount of information shared by them.

Another interesting measure is the conditional cross entropy between X and Y given a non empty subset S. Generally speaking, for three random variables X, Y and Z, we have the following definition

Definition 3.4 (Conditional cross entropy)**.** The conditional cross entropy (CCE for brevity) between two random variables X and Y given Z is:

$$CE(X, Y|Z) = \sum_{z} P(z) \sum_{x,y} P(x, y|z) \cdot log \frac{P(x, y|z)}{P(x|z)P(y|z)}. \tag{3.3}$$

The $CE(X, Y|Z)$ can be also formulated as:

$$CE(X, Y|Z) = H(X|Z) - H(X|Y, Z)$$

where $H(X|Z)$ is the conditional entropy of the variable pair X and Y and $H(X|Y, Z)$ is the entropy of X given the subset (Y, Z). The conditional cross entropy is equal to zero when X and Y are conditionally independent given Z.
Given a set of data $\mathcal{D}$, we can test whether X and Y are conditional independent given a set of variables S computing the conditional cross entropy $CE(X, Y|S)$. To test the degree of (conditional) independence of two or more variables, PC algorithm uses the $G^2(X, Y, S)$ statistic which is $2nCE(X, Y|S)$ where n is the sample size. Under the null hypothesis of independence, G^2 follows a χ^2 distribution (Lindgren 1976) with degrees of freedom equal to $(k_x - 1)(k_y - 1) \prod_{z \in S} k_z$ where k_x, k_y, k_z respectively denote the number of values of variables X, Y and $Z \in S$. Given a significance level α:

- if $G^2(X, Y, \varnothing) < \chi^2_{(1-\alpha, df)}$ then X and Y are marginally independent and we write $X \perp\!\!\!\perp Y$;
- if $G^2(X, Y, S) < \chi^2_{(1-\alpha, df)}$ then X and Y are conditionally independent given S and we write $X \perp\!\!\!\perp Y|S$.

3.1.2 Algorithm structure

PC algorithm is a constraint-based algorithm to learn the DAG structure from data. It takes observed data $\mathcal{D}$ stored in a dataset as input and it provides a DAG G equivalent to the true one. Advantages of learning the

Markov equivalence class represented by PDAG instead of the DAG have been discussed in the literature (Chickering 2002, e.g.).
Given a set of K measured variables $\boldsymbol{X} = (X_1, X_2, ..., X_K)$ the procedure starts by drawing a complete undirected graph G' with K nodes, representing variables. At each iteration, the PC algorithm tests independences between a pair of variables $\in \boldsymbol{X}$, say X_a and X_b, given a non empty subset of variables $S \in \boldsymbol{X} \setminus \{X_a, X_b\}$ and it deletes recursively edge between nodes a and b if X_a and X_b are conditionally independent given S.
The PC algorithm represents an improved version of two previous constraint-based algorithms, namely *SGS* and *IC* (Pearl 2000). These algorithms search, step by step, all possible subsets of $V \setminus \{a, b\}$ such that $X_a \perp\!\!\!\perp X_b | S$. PC algorithm, instead, considers in S only variables corresponding to nodes belonging to the set of adjacent nodes of a, denoted by $ne(a)$, at the current iteration. The computational advantage of this, consists in limiting the number of performed statistical tests.
The structure of PC algorithm consists in three main steps:

1. Find the skeleton of the graph;

2. find the head-to-head configurations;

3. orient the rest of the links without producing any cycle and any other head-to-head configuration.

Find the skeleton of the graph.

Let $\boldsymbol{X}$ be a set of K random variables. Let V be a set of K nodes in a graph so that each node in V represents a random variable in $\boldsymbol{X}$. For each pair of ordered nodes in the graph, conditional independence between the corresponding variables is tested according to the procedure in Algorithm 1.

The first step of the algorithm starts with a complete undirected graph, here denoted by G', that is a graph where all nodes are connected to each other. An example of complete undirected graph on 5 nodes is provided in Figure 3.1. The Figure 3.1 shows a complete undirected graph where node

Algorithm 1 PC algorithm: Find skeleton

```
Start with a complete undirected graph $G'$
$\ell = 0$
repeat
  for each $a \in V$ do
    for each $b \in ne(a)$ do
      Test whether $\exists S \in ne(a) \setminus \{b\}$ with $|S| = \ell$ and $X_a \perp\!\!\!\perp X_b | S$
      if this set exists then
        Make $S_{ab} = S$
        Remove link between $a$ and $b$ from $G'$
      end if
    end for
  end for
  $\ell = \ell + 1$
until $|ne(a)| \leq \ell \; \forall a$
```

labels coincide with the names of the variables.

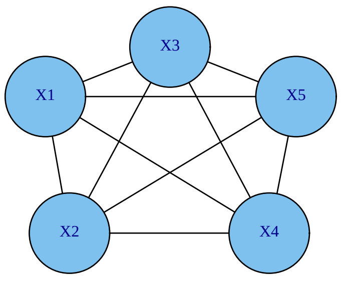
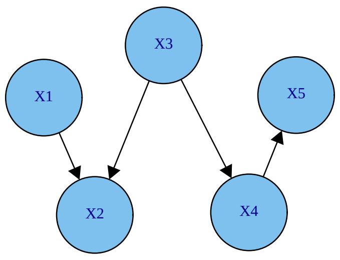


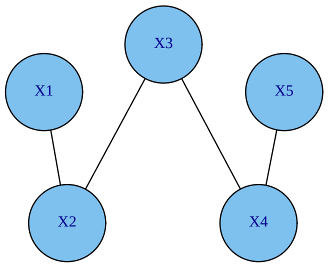
Figure 3.1: A complete undirected graph

At the first iteration, the cardinality ℓ of the subset S is set equal to zero, so the marginal independence between each pair of variables is tested. For a generic pair of variables, X_a and X_b, if they are independent, the edge between a and b is removed; if X_a and X_b are dependent the edge between a and b is maintained. After having checked all marginal independences, the cardinality of the subset S is increased by one. Given $\ell = 1$, the procedure tests the independence between each pair of adjacent nodes given another node in $ne(a)$. Also in this iteration, if the generic pair of variables X_a and X_b are independent given S the edge between node a and node b is deleted,

otherwise is maintained. This mechanism is repeated at every iterations. The procedure stops when the number of nodes adjacent to a is less or equal to the cardinality of S.
The output of this first step is the underlying undirected graph, also called skeleton of the graph, that is the DAG basic structure that ignores the directions. An example of DAG and its skeleton is displayed in the Figure 3.2 below.



(a) A DAG



(b) The skeleton

Figure 3.2: A DAG and its skeleton

Find the head-to-head configurations.

A head-to-head configuration has been introduced in Section 2.3. It is a converging configuration, also called v-structure, containing a *collider* node. For instance, Figure 3.3 shows a graph with three nodes called X_1, X_2 and X_3. Since X_1 and X_3 are not connected to each other while an arrow exists both between X_1 and X_2 and between X_3 and X_2, X_2 is a collider node.

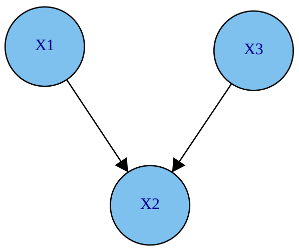


Figure 3.3: A collider

A head-to-head configuration encodes a conditional dependence relation. For instance, according to the graph in Figure 3.3, X_1 and X_3 are not conditionally independent given X_2.
The procedure to find the head-to-head configurations is in Algorithm 2.

Algorithm 2 PC algorithm: head-to-head configuration

```
for each $a - c - b$ structure do
  if $c \notin S_{ab}$ then
    Orient $a - c - b$ as $a \rightarrow c \leftarrow b$
  end if
end for
```

So, according to the algorithm, if two generic variables X_a and X_b are not conditionally independent given a subset $S_{ab} = X_c$ than X_c is a collider and a v-structure $a \rightarrow c \leftarrow b$ exists; if X_a and X_b are conditionally independent given a subset $S_{ab} = X_c$ then X_c is not a collider and, at this stage, edges remain undirected $a - c - b$.

Orient the rest of links without producing any cycle and any other head-to-head configurations.

In the last step of the algorithm some constraints must be fulfilled: no new head-to-head configurations can be created; cycles in the graph are forbidden. For these reasons some rules for properly orienting edges exist and are encoded in Algorithm 3.

At the end of this step some edges can remain undirected. The output of the third step of the algorithm is a PDAG, that represents the equivalence class of those graphs Markov equivalent to the true unknown DAG.

A version of the PC-algorithm is implemented in the `R`-package `pcalg`. `R` is an open source software consisting in many packages. Both `R` and some packages can be downloaded from `http://www.r-project.org`. The PC algorithm is implemented in the following others software:

- Hugin (available at `http://www.hugin.com`),

Algorithm 3 PC algorithm: edges orientation

```
while more edges can be oriented do
  for each $a \rightarrow c - b$ structure do
    Orient $c - b$ as $c \rightarrow b$
  end for
  for each $a - b$ such that there is a directed path from $a$ to $b$ do
    Orient $a - b$ as $a \rightarrow b$
  end for
  for each $a - c - b$ structure such that $a \rightarrow w$;$b \rightarrow w$;$c - w$ do
    Orient $c - w$ as $c \rightarrow w$
  end for
end while
```

- Murphys Bayes Network toolbox (at `http://bnt.sourceforge.net`),
- Tetrad IV (at `http://www.phil.cmu.edu/projects/tetrad`).

3.1.3 Some variations on the PC algorithm

As previously said, the assumptions under which PC algorithm correctness is proved, are often not verified. Over recent years, many other limitations have been pointed out from different authors and some variations on PC algorithm have been proposed in order to overcome the highlighted problems.
One of the best variation is the *NPC algorithm* (Steck 2001). This procedure seeks to solve PC algorithm problems coming from limited dataset. The basic mechanism is the same of PC algorithm but it introduces two innovations: the *necessary path condition* criterion, from whom the algorithm takes the name, and *ambiguous regions* concept. Given two variables in the graph X_a and X_b, the necessary path condition criterion suggests to consider, as possible conditioning set, only variables falling in an undirected path between X_a and X_b. This allows to reduce the number of independence tests to conduct and, therefore, possible mistakes. Errors can be further reduced using the notion of ambiguous regions. Roughly speaking, if the presence of an edge e_1 is dependent on the absence of another edge e_2, e_1 and e_2 are said to be inter-dependent. In this situation PC algorithm would maintain/remove the edge firstly tested. NPC algorithm, instead, does not decide immediately

between e_1 or e_2 but treats both of them as uncertain links. All uncertain links found by NPC constitute an ambiguous region. To solve ambiguities, the NPC algorithm relies on the user interaction who has the opportunity to decide on addition, removal or reversal direction of arcs using his subject matter knowledge. As a consequence of this, NPC algorithm result does not depend on the sequence in which test are conducted.
Another discussed limitation of PC algorithm deals with the fact it does not possess a mechanism either to consider or to treat mistakes in independence tests. Whit respect to this aspect, Fernandes *et al.* (2004) proposed an extended version of the PC algorithm, namely *XPC* algorithm. Since it combines some techniques of PC algorithm and some elements of NPC algorithm, it represents a synthesis of PC and NPC algorithms capable to reduce the number of structural errors. XPC algorithm has the following procedure: it tests marginal and conditional independence by means of MDL measure, i.e. on the basis of penalized maximum likelihood; it reduces the number of tests considering only the necessary path condition; it identifies ambiguous regions that will be solved by experts.
Many other procedures and combination of techniques have been developed in the literature (Abellán *et al.* 2006). However, up to now nobody focused on the fact that PC algorithm does not distinguish among categorical variables and it handles ordinal variables as nominal variables when conditional independence tests are conducted. In Sections 3.3 and 3.4 two variations on PC algorithm are proposed. The first one, namely *OPC algorithm*, is useful when ordinal variables occur; the second one, namely *NOPC algorithm*, is appropriate in presence of mixed nominal and ordinal categorical variables. Both variations differ from PC algorithm in the test used for checking conditional independence statements. For this reason, some appropriate nonparametric tests for ordinal and nominal-ordinal variables have been selected. We selected nonparametric methods for several reasons:

- they do not require stringent assumptions about the population distribution from which sample are drown;

- they often are the most appropriate for categorical data expressing order or counts of numbers in categories (Sprent and Smeeton 2001).

Further features of nonparametric tests and the illustration of selected tests are discussed in Section 3.2.

3.2 Rank-based tests

In some studies such as behavioral, marketing, evaluation and medical surveys, researchers are interested in testing hypotheses coming from specific theories. Hypothesis tests are statistical techniques that help researchers to this aim. After having built the hypotheses set, data are collected and are used to decide about either accepting or rejecting a certain hypothesis. The taken decision leads the researcher to understand if a specific theory is confirmed or not by the set of data. Parametric techniques are usually the first approach adopted in this kind of statistical analysis. However, since they require sine assumptions, they may be difficult to apply. For these reasons, techniques that do not require numerous or stringent assumptions have been developed. These methods are known as *nonparametric* and they mainly consist in a set of statistical tests. Nonparametric tests are often considered as not systematic and wasteful, i.e. less efficient than parametric tests. On the other hand, there are many advantages for adopting nonparametric methods:

- they do not need assumptions about the distribution from which data have been sampled (this is the reason why they are called "distribution-free");
- they are the only available for data expressing ranking;
- they are really useful when the sample size is small.

Pros and cons of nonparametric tests have been largely discussed by Siegel and Castellan (1988).
Nonparametric tests can deal with different problems such as goodness of fit, homogeneity, symmetry and independence. Here we consider tests of

independence, or more specifically, of homogeneity[1]. These tests are often based on the notion of *rank*. In detail, ranks and order statistics are basic tools of nonparametric methods, so they are briefly introduced below.
Let $X_1, X_2, ..., X_n$ be a random sample with cumulative distribution function F. Let $X^{(i)}$ be the i-th smallest observation in $X_1, X_2, ..., X_n$. Then $X^{(i)}$ is a statistic called *i-th order statistics*. The first order statistic is the minimum value of the sample and the last order statistic is the maximum value of the sample so that $X^{(1)} \leq X^{(2)} \leq ... \leq X^{(n)}$ is the sequence obtained ordering observations in a non decreasing order. The vector of order statistics is usually denoted by $X^{(\cdot)} = (X^{(1)}, X^{(2)}, ..., X^{(n)})$. In the ordered sample, the place occupied by X_i is said *rank* of X_i and is denoted by R_i. Roughly speaking, ranks are serial numbers of observations arranged in an increasing order. Specifically, the statistic R_i, rank of X_i, is the number of observations **smaller then or equal to** X_i. The vector of ranks is usually denoted by $R = (R_1, R_2, ..., R_n)$.

Definition 3.5 (Ranks)**.** Let $X_1, X_2, ..., X_n$ be a random sample with cumulative distribution function F. Let $I(X_i \geq X_j)$ be the following indicator function:

$$I(X_i \geq X_j) = \begin{cases} 1 & X_i \geq X_j \\ 0 & X_i < X_j \end{cases}$$

Ranks are statistics defined by the following transformation:

$$R_i = \sum_{j=1}^{n} I(X_i \geq X_j),\ i = 1, 2, ..., n \tag{3.5}$$

[1]Two generic random variable X and Y are said to be independent if their joint distribution can be factorized as follows:

$$P(X = x, Y = y) = P(X = x)P(Y = y)$$

As a consequence of this the conditional probability of Y given $X = x$ is the following:

$$P(Y = y|X = x) = P(Y = y) \tag{3.4}$$

When X and Y are independent the probability of Y does not involve the probability of X; if the (3.4) hold for each level of X, it is possible speaking of homogeneity rather than independence. This consideration can be extended for conditional independence between two discrete variables given a third discrete variable.

The main idea of rank-based tests is to build a test based on a statistic that is function of ranks only.
The theory of rank tests is usually based on the assumption that observed variables come from a continuous distribution. This assumption ensures all observations are distinct and ranks are well defined. However, in observational studies, measured variables are often discrete so observations with the same magnitude, namely *ties*, can occur. In this case, ranks are not well defined and it is necessary to use a method for dealing with ties. The most frequently applied techniques for treating ties consists in computing an average of the ranks which would have been assigned to tied. This mean values are called *midranks*; here the vector of midranks is denoted by r. Suppose to have n distinct observations on two variables, X with 2 levels and Y with C levels. Data with ties can be arranged as in Table 3.1.

	Y				
X	$\tilde{y}^{(1)}$	$\tilde{y}^{(2)}$	....	$\tilde{y}^{(C)}$	Total
1	n_{11}	n_{12}	....	n_{1C}	n_{1+}
2	n_{21}	n_{22}	....	n_{2C}	n_{2+}
Total	n_{+1}	n_{+2}	....	n_{+C}	n

Table 3.1: Data with ties arranged in a table

In the Table 3.1, the columns $\tilde{y}^{(1)}, \tilde{y}^{(2)}, ..., \tilde{y}^{(C)}$ are the C distinct order statistics of Y. Since Y is discrete, the vector $\tilde{y}^{(1)}, \tilde{y}^{(2)}, ..., \tilde{y}^{(C)}$ is simply the vector of the C levels of Y. The value in the generic cell n_{ij} represents the number of observations in i level of X with $Y = \tilde{y}^{(j)}$. For data arranged as in Table 3.1 the generic midrank r_j is the average between ranks $\tau + 1$ and $\tau + n_{+k}$, that is :

$$r_j = \tau + \frac{n_{+j} + 1}{2}$$

where

$$\tau = \sum_{l=1}^{j-1} n_{+l}.$$

In the following some rank-based tests are introduced. Specifically, the Wilcoxon test is introduced firstly when distinct observations occur and then for data arranged in tied form. Other nonparametric tests discussed in this work are presented for data arranged in tied form only. We will use the notation of Edwards (1995).

Wilcoxon test

It is often interesting to know if two random variables, say X and Y, have the same cumulative distribution function (homogeneity between X and Y distributions) or not. nonparametric tests checking this aspect are known as test of homogeneity. One of the best known and probably most frequently used among the two-sample rank tests is the *Wilcoxon* test. The Wilcoxon test compares the null hypothesis of homogeneity against the alternative hypothesis of no homogeneity between X and Y distributions and it makes decision by using a statistic proposed by Wilcoxon (1945).
Let $X_1, X_2, ..., X_{n_1}$ be an independent and identically distributed random sample, from now on *iid*, and let F be the cumulative distribution function of X. Let $Y_1, Y_2, ..., Y_{n_2}$ be an *iid* random sample and let G be the cumulative distribution function of Y. The null hypothesis of homogeneity can be expressed as follows:

$$H_0 : F \equiv G$$

The alternative hypothesis can be formulated in different ways:

$$H_1 : F \not\equiv G$$

that means there is not homogeneity between X and Y;

$$H_1 : F > G, \textit{ or}$$
$$H_1 : F < G$$

meaning that X stochastically dominates Y and that X is stochastically dominated by Y respectively.
The test statistic used is given by the sum of observation ranks in the pooled sample. Consider $n = n_1 + n_2$ so that the sample $X_1, X_2, ..., X_{n_1}, Y_1, Y_2, ..., Y_{n_2}$ can be considered as a random sample. Let $R_1, R_2, ..., R_{n_1}$ be the ranks of $X_1, X_2, ..., X_{n_1}$ and let $R_{n_1+1}, R_{n_1+2}, ..., R_{n_2}$ be the ranks of $Y_1, Y_2, ..., Y_{n_2}$. The Wilcoxon statistic is the following:

$$W = \sum_{i=1}^{n_1} R_i \tag{3.6}$$

The statistic (3.6) gives the rank sum of $(X_1, X_2, ..., X_{n_1})$. It can be expressed in different ways. An alternative way to write (3.6) is by denoting:

$$C(i) = \begin{cases} 1 & i = 1, ..., n_1 \\ 0 & i = n_{1+1}, ..., n \end{cases}$$

and

$$W = \sum_{i=1}^{n} C(i) R_i.$$

Another form of (3.6) is due to Mann and Whitney (1947). This form, not referring to stochastic ordering, is based on the statistic test U that is the number of pairs X_i, X_j with $X_i < X_j$, $i = 1, 2, ..., n_1, j = 1, 2, ..., n_2$.

$$U = n_1 n_2 + \frac{n_1(n_1+1)}{2} - W$$

Consider now the case of n observations on three discrete variables X, Y and S with $T = 2$, C and L levels respectively. Suppose we are interested in computing a test for $X \perp\!\!\!\perp Y|S$. In this situation, data are arranged in a $T \times C \times L$ three way table. Tied observations occur therefore the Wilcoxon statistic expression in (3.6) is not appropriate. The $T \times C \times L$ three way table can be alternatively expressed as L different $T \times C$ tables. For a fixed level k of S, $k = 1, ..., L$, a slice of the $T \times C \times L$ tables is Table 3.2.

	Y				
X	1	2	....	C	Total
1	n_{11k}	n_{12k}	....	n_{1Ck}	n_{1+k}
2	n_{21k}	n_{22k}	....	n_{2Ck}	n_{2+k}
Total	n_{+1k}	n_{+2k}	....	n_{+Ck}	n_{++k}

Table 3.2: A slice of the $2 \times C \times L$ table

Suppose the distribution of Y given $X = i$ and $S = k$ is denoted by $F_{i,k}(y)$. The hypothesis set of Wilcoxon test can be written as follow:

$$\begin{aligned}
&H_0 : F_{1,k}(y) = F_{2,k}(y),\ \forall y, \forall k \\
&H_1 : F_{1,k}(y) \neq F_{2,k}(y), \forall y \text{ and for some } k
\end{aligned}$$

The test statistic W is the sum of the ranks of the first group over all strata, that is:

$$W = \sum_{k=1}^{L} R_{1k}$$

We generally denote by R_{ik} the rank sum for the i-th level of X in the k-th stratum, that is:

$$R_{ik} = \sum_{j=1}^{C} r_{jk} n_{ijk} \tag{3.7}$$

In the formula (3.7), r_{jk} is the midrank of an observation of column j of Y in stratum k of S.
Under the null hypothesis, in the conditional distribution, W has the following mean:

$$E(W|H_0) = \sum_{k=1}^{L} \frac{n_{1+k}}{n_{++k}} \sum_{j=1}^{C} r_{jk} n_{+jk}$$

and the following variance:

$$\hat{Var}(W|H_0) = \sum_{k=1}^{L} \frac{n_{1+k} n_{2+k}}{n_{++k}(n_{++k}-1)} \sum_{j=1}^{C} \left[r_{jk} - \frac{E(R_{1k}|H_0)}{n_{1+k}} \right]^2 n_{+jk}$$

Asymptotically and under H_0, W posses a normal distribution and we have:

$$\frac{W - E(W|H_0)}{\sqrt{\hat{Var}(W|H_0)}} \sim N(0,1)$$

and the two-sided p-value can be computed as:

$$p = P(|W - E(W|H_0)| \geq |W_{obs} - E(W|H_0)||H_0).$$

Here, the Wilcoxon test is used for testing homogeneity between a row binary variable and a column discrete (nominal or ordinal) variable.

Kruskall-Wallis Test

The Kruskal-Wallis test was introduced by Kruskall (1952) and Kruskall and Wallis (1952) as a tool for testing the null hypothesis of homogeneity among several distributions (more than two), against the alternatives hypothesis of no homogeneity for at least two distributions.

Consider two variables X and Y with $T > 2$ and C levels respectively. Suppose we are interested in computing conditional test for $X \perp\!\!\!\perp Y|S$ where S is a conditioning variable with L levels. Suppose for the k-th level of S, data are organized as in Table 3.3.

	Y				
X	1	2	....	C	Total
1	n_{11k}	n_{12k}	....	n_{1Ck}	n_{1+k}
2	n_{21k}	n_{22k}	....	n_{2Ck}	n_{2+k}
....	....	....	....	....	....
T	n_{T1k}	n_{T2k}	....	n_{TCk}	n_{T+k}
Total	n_{+1k}	n_{+2k}	....	n_{+Ck}	n_{++k}

Table 3.3: A slice of the $T \times C \times L$ table

Let $F_{i,k}(y)$ denote the distribution of Y given $X = i$ and $S = k$. The null hypothesis of homogeneity and the alternative one can be written as follows:

$$H_0 : F_{1,k}(y) = F_{2,k}(y) = ... = F_{T,k}(y), \ \forall y, \forall k, \ or$$
$$H_1 : F_{i,k}(y) < F_{j,k}(y), \forall y \text{ for some } k \text{ and } i \neq j$$

The alternative hypothesis suggests that there is at least one distribution that dominates stochastically another distribution.
The test statistic used is:

$$KW = \sum_{k=1}^{L} f_k \sum_{i=1}^{T} \frac{\left[R_{ik} - \frac{n_{i+k}(n_{++k}+1)}{2}\right]^2}{n_{i+k}}$$

where

$$f_k = 12 \left\{ n_{++k}(n_{++k}+1) \left[1 - \frac{\sum_{j=1}^{C}(n_{+jk}^3 - n_{+jk})}{n_{++k}^3 - n_{++k}} \right] \right\}^{-1}$$

and R_{ik} is the rank sum for row i and stratum k.
Under the null hypothesis the Kruskal-Wallis test statistic is asymptotically χ^2 distributed with $L(T-1)$ degrees of freedom.
Here, the Kruskall-Wallis test is used for comparing a row nominal variable with a column discrete (nominal or ordinal).

Jonckheere-Terpstra Test

The Jonckheere Terpstra test was proposed independently and quite contemporary by Terpstra (1952) and Jonckheere (1954) as a nonparametric test for trend among ordered alternatives. It is similar to the Kruskal-Wallis test but it has a more specific alternative hypothesis. When researchers need to entertain a more specific alternative hypothesis in the problem formulation, the Jonckheere-Terpstra test can be useful. Generally speaking, given two *iid* samples, the test compares the null hypothesis of no systematic trend across samples against the alternative that samples are ordered in a specif *a priori* sequence. Jonckheere-Terpstra test statistic counts the number of times an observation of the i-th sample is preceded by an observation of j-th sample. This amount is the Mann-Whitney statistic, so Jonckheere-Terpstra test is the sum of certain Mann-Whitney statistics (Hollander and Wolfe 1999).

Consider now data are n observations on three variables X, Y and S. Suppose we are interested in checking $X \perp\!\!\!\perp Y|S$ where X and Y are both ordinal with T and C levels respectively and S is a discrete variables with L levels. Let $F_{i,k}(y)$ be the distribution of Y given $X = i$ and $S = k$. The null hypothesis of Jonckheere-Terpstra test is that of homogeneity, that is:

$$H_0 : F_{1,k}(y) = F_{2,k}(y) = ... = F_{T,k}(y), \forall y, \forall k$$

This is tested against the alternative hypothesis of a stochastic ordering among distributions.

$$\begin{aligned} &F_{i,k}(y) > F_{j,k}(y), \text{ with } i < j, \forall y, \forall k, \textit{ or} \\ &F_{i,k}(y) < F_{j,k}(y), \text{ with } i > j, \forall y, \forall k \end{aligned}$$

The test statistic is:

$$JT = \sum_{k=1}^{L} \sum_{i=2}^{T} \sum_{j=1}^{C-1} \left\{ \sum_{s=1}^{C} w_{ijsk} n_{isk} - \frac{n_{i+k}(n_{i+k}+1)}{2} \right\}$$

where w_{ijsk} are Wilcoxon scores denoted by:

$$w_{ijsk} = \sum_{t=1}^{s-1} (n_{itk} + n_{jtk}) + \frac{(n_{isk} + n_{jsk} + 1)}{2}$$

Under the null hypothesis the mean of JT is the following:

$$E(JT|H_0) = \frac{\sum_{k=1}^{L}(n_{++k}^2 - \sum_{i=1}^{T} n_{i+k}^2)}{4}$$

The asymptotic variance, discussed by Lehmann (1975) and Pirie (1983), is:

$$V\hat{a}r(JT|H_0) = \frac{V_1}{72} + \frac{V_2}{36(n_{++k}(n_{++k}-1)(n_{++k}-2))} + \frac{V_3}{8(n_{++k}(n_{++k}-1))}$$

where:

$$\begin{aligned}
V_1 &= n_{++k}(n_{++k}-1)(2n_{++k}+5) - \sum_{i=1}^{T} n_{i+k}(n_{i+k}-1)(2n_{i+k}+5) - \\
&\quad - \sum_{j=1}^{C} n_{+jk}(n_{+jk}-1)(2n_{+jk}+5) \\
V_2 &= \sum_{i=1}^{T} n_{i+k}(n_{i+k}-1)(n_{i+k}-2) - \sum_{j=1}^{C} n_{+jk}(n_{+jk}-1)(n_{+jk}-2) \\
V_3 &= \sum_{i=1}^{T} n_{i+k}(n_{i+k}-1) - \sum_{j=1}^{C} n_{+jk}(n_{+jk}-1)
\end{aligned}$$

Asymptotically, the test statistic is a standard normal and the two-sided p-value is given by:

$$p = Pr(|JT - E(JT|H_0)| \geq |JT_{obs} - E(JT|H_0)||H_0)$$

Here, the Jonckheere-Terpstra test is used for comparing a row ordered variable with a column ordered variable. In detail, while the Wilcoxon and Kruskall-Wallis tests have been used in NOPC algorithm only, the Jonckheere-Terpstra test has been added in both PC and NOPC algorithm procedures.

3.3 OPC algorithm

Having recalled some nonparametric methods useful for this thesis, it is now possible to introduce the new algorithms.
The Ordinal PC algorithm, *OPC algorithm*, is a variation on PC algorithm capable to consider information provided by ordinal variables. PC algorithm, in fact, is not able to consider variables class and it treats categorical variables as nominal even if they are ordinal. This can produce information loss. It would be worth to learn Bayesian network with a constraint-based approach without demoting ordinal variables in nominal variables. OPC algorithm represents the first attempt in this direction. A previous application of

graphical models to ordinal variables (Stanghellini 1999) do not manage the problem according to the same approach used here.
The OPC algorithm structure is the same of PC algorithm. It is still a three steps algorithm:

1. it starts with a complete undirected graph; it firstly checks checking marginal independences and then conditional independences between variables until the underlying graph is found;

2. it looks for head-to-head configurations;

3. it orients the rest of links without producing any cycle and any new head-to-head configuration.

The difference between PC and OPC algorithm is in the statistical procedure, i.e. the test used for checking conditional independences. Ordinal variables have a natural ordering among categories but distance between categories is unknown (see Section 1.1), so data have a ranking but not a numerical interpretation. In these cases nonparametric methods are necessary. For testing independence between two ordinal variables a rank-based test is required. For this reasons, we use the Jonckheere-Terpstra test in the OPC algorithm procedure. More specifically, the edge between two nodes in the graph is removed if the p-value of Jonckheere-Terpstra test between variables is less than a fixed significance level α. The procedure is available in Appendix A. The function `OPC` has been written following the same scheme of `pcAlgo` and substituting the original independence test with the Jonckheere-Terpstra test. The test is implemented in *MIM* (Edwards 1995) a software for graphical modelling. In MIM, the test is useful to check the conditional independences for pairs of ordinal variables and can be used in the stepwise model selection procedure. In order to add the test in the OPC procedure, an `R` function that supports Jonckheere-Terpstra test, has been written. This function, namely `CI.discrete` (see Appendix A for further details), is the result of a joint work with D. Edwards. It checks conditional independences for discrete variables (see Section 3.2) by previously discussed exact tests.

3.4 NOPC algorithm

OPC algorithm is useful for learning a Bayesian network from ordinal data. OPC algorithm accuracy is particularly appropriate when a monotone association between variables can be reasonably assumed. Let X and Y be two ordinal variables. A monotonic increasing (decreasing) trend exists if for large values of X, large (small) values of Y are observed. A way to measure the monotone association is by means of metrics based on concordance and discordance (Agresti 2010). In Section 1.2 concordant, discordant and tied pair have been introduced. Here, we provide an example. Consider the following contingency table of simulated data where X is an ordinal variable with three levels and Y is an ordinal variable with two levels.

	Y	
X	y_1	y_2
x_1	15	20
x_2	10	50
x_3	4	40

Table 3.4: A contingency table for two ordinal variables X and Y monotonically associated

For a pair of units in the cell $(x_1; y_1)$ and $(x_2; y_2)$ the second subject ranks higher than first both in X and Y. This means that the pair is concordant. This occurs whenever a subject in $(x_1; y_1)$ is compared with a subject in $(x_2; y_2)$. Each subject in $(x_1; y_1)$ forms a concordant pair also with subjects in $(x_2; y_3)$. So subjects who rank higher than $(x_1; y_1)$ both on X and Y are $15 \cdot (50 + 40)$. There are also $10 \cdot 40$ subjects in $(x_3; y_2)$ that rank higher than $(x_2; y_1)$ on both X an Y. The total number of concordant pairs in observations is denoted by C. In this example $C = 1200$.
For a pair of subjects in the cells $(x_1; y_2)$ and $(x_2; y_1)$ the second subject is higher than the first in X but it is lower than the first in Y. The pair is discordant as also the pair of subjects in the cells $(x_1; y_2)$ and $(x_3; y_1)$. These occur $20 \cdot (10 + 4)$ times and $50 \cdot 4$ times respectively. The total number of discordant pairs in observations is denoted by D. In this example $D = 450$

and $C > D$. This suggests a tendency for those having higher levels of X to be grater in Y too. We introduce the *Gamma* index in Section 1.2. This is an association measure based on concordances an discordances. For more details about the measure, see Goodman and Kruskal (1979). Let $\prod_c$ and $\prod_d$ be the probabilities of concordance and discordance of a pair of independent observations from a joint distribution π_{ij}.

$$\prod_c = 2\sum_i \sum_j \pi_{ij}(\sum_{h>i} \sum_{k>j} \pi_{hk})$$
$$\prod_d = 2\sum_i \sum_j \pi_{ij}(\sum_{h>i} \sum_{k<j} \pi_{hk})$$

where $(i; j)$ and $(h; k)$ are cells in a contingency table.
Gamma is a measure based on the distance between concordant and discordant probabilities:

$$\gamma = \frac{\prod_c - \prod_d}{\prod_c + \prod_d}$$

The sample version of *Gamma* measure is:

$$\hat{\gamma} = \frac{C - D}{C + D} \tag{3.8}$$

It has range $-1 \leq \gamma \leq 1$. For the data in our example $\gamma = 0.45$. This suggests a monotone increasing association between X and Y in the table. Since a monotone trend exists the Jonckheere-Terpstra test is appropriate. The p-value associated with the JT test statistic is close to 0. This strongly suggests to reject the null hypothesis.

However when the initial assumption of presence of monotone trend is not verified, Jonckheere-Terpstra test may fail. Consider a third level of Y that changes the association between X and Y. Data are displayed in the Table 3.5.

The number of concordant pairs, 3225, is very close to the number of discordant pairs, 3220, so γ is almost zero. In this case data strongly

	Y		
X	y_1	y_2	y_3
x_1	15	20	20
x_2	10	50	15
x_3	4	40	10

Table 3.5: A contingency table for two ordinal variables X and Y non monotonically associated

suggest there is no evidence of monotonic association between X and Y. In this example the Jonckheere-Terpstra test is not appropriate since there is not a monotone trend. It gives a p-value equal to 0.99 suggesting there is no evidence in data to refuse null hypothesis. However, $\gamma = 0$ not necessarely implies independence. A type of association between variables still exists and a different test would be able to catch it. The p-value associated with the G^2 test statistic, for instance, is equal to 0.000 and it strongly suggests to reject H_0. This example leads to the consideration that a preventive analysis of contingency tables can be useful for choosing the most appropriate test for checking conditional independence. A test appropriate for a pair of variables could be non appropriate for another pair of variables. This can happen when mixed nominal-ordinal data are observed. In observational studies, for instance, both socio-demographic features and opinions can be measured. In the first case variables are mainly nominal, while opinions can be taken over ordinal scales. In this case, neither PC nor OPC algorithm are appropriate. As previously said PC algorithm is not able to take into account information provided by ordinal data; on the other hand, OPC algorithm is suitable for ordinal data only. So, a constraint-based algorithm capable to treat each possible pair of categorical variables could be useful. Nominal-Ordinal PC algorithm, *NOPC algorithm*, is a proposal for solving this problem. The algorithm is capable to choose the right test according to each variable pair. In detail, it is firstly necessary to set the class of each categorical variable: nominal or ordinal. A specific function useful for this purpose is available in Appendix A. After having set the class of variables, the test for checking conditional independence for discrete variables is automatically selected among those supported in `CI.discrete` function. The procedure selects automatically one test among Wilcoxon, Kruskall-Wallis and Jonckheere-Terpstra test. In detail, if variables are both nominal, the conditional cross entropy, CCE, is computed; the Wilcoxon test is adopted if one variable is ordinal and one variable is binary; the Kruskall-Wallis test is chosen if one variable is ordinal and the other is nominal; when variables are both ordinal, the Jonckheere-Terpstra test is selected. The table 3.6 summarises tests automatically selected for each categorical variable pair.

	Nominal	**Binary**	**Ordinal**
Nominal	CCE	CCE	Kruskal-Wallis
Binary	CCE	CCE	Wilcoxon
Ordinal	Kruskal-Wallis	Wilcoxon	Jonckheere-Terpstra

Table 3.6: Available tests in CI.discrete function

It is worth noting that the NOPC result depends on the sequence in which tests are carried out and on the variable class. This is the reason why a pre-ordering among variables in dataset, coherent with knowledge about phenomenon and according to logical meaning of variables, can be useful to improve the performance of the algorithm. In addition, setting the variable class it is possible to take into account information gained studying the contingency tables: for instance if there is not a monotone association between two ordinal variables it can be more relevant to treat them as nominal.

In this work the performance of NOPC algorithm are not going to be tested but the procedure is available in Appendix A. The next Chapter deals with OPC algorithm performance: two simulation studies are presented and some results are discussed.

Chapter 4

Applications and results

In Chapter 3 the structural learning and its approaches have been discussed. In Section 3.1 the PC algorithm, one of the most used constraint-based algorithms, has been presented and some limitations have been pointed out. One of these is that the PC algorithm handles all categorical variables as nominal. In some contexts, such as biostatistics, evaluation, marketing or psychological surveys, features are measured on ordered categories, so they are ordinal variables. Ignoring the ordering among levels of these variables can produce a loss of information that can affect the multivariate analysis of data and, in graphical models framework, the structural learning. For this reason a PC algorithm variation called OPC algorithm has been proposed in Section 3.3. The new algorithm is able to perform the structural learning taking into account information provided by ordinal variables; this is possible since, step by step, the procedure checks associations and, thus, decides about edges presence using a nonparametric rank-based test (see Section 3.2). However, the OPC algorithm is not able to handle mixed nominal and ordinal variables, so a new variation called NOPC algorithm has been proposed (see Section 3.4). The NOPC algorithm is able to automatically select, according to the type of variables, the most appropriate nonparametric test for checking independences.
In this Chapter, some applications of the OPC algorithm on ordinal data are going to be presented. In order to evaluate the goodness of the OPC learning

and to compare the OPC and PC algorithm, some performance indicators are introduced (see Section 4.1). In literature, several evaluation metrics are available: they usually compare the true models structure, supposed to be known, with that learnt from data. The common procedure to evaluate algorithm performance consists in generating a training set of data according to the true structure and in performing the structural learning on these data. The simulation studies we are going to present follows these steps:

- in `Hugin` a network is learnt in from a dataset using the PC algorithm and the estimated DAG is supposed to be the true structure;
- on the basis of the true DAG several datasets (1000) have been generated for different sample sizes (50, 100, 500);
- on simulated data, the structural learning has been performed using both PC and OPC algorithm;
- the algorithms performance has been measured comparing the true model the model learnt from generated data;
- results of the applications and a comparison between OPC and PC performance are presented in Section 4.2.

We performed simulations in `R` following the code available in Appendix B.

4.1 Performance measures

The OPC algorithm has been proposed with the aim to increase the PC algorithm performance when ordinal variables are observed. Algorithm performance can be measured in term of completeness, reliability and accuracy so, several different performance indicators can be used. Some researchers measure the performance comparing the skeletons of both learnt and true DAGs (Kalisch and Bühlmann 2007; Li and Wang 2009); others calculate the structural distance between DAGs (Monti and Cooper 1997; Cooper and Herskovits 1992; Cruz-Ramírez *et al.* 2006) or equivalence classes (Cano *et al.* 2008; Abellán *et al.* 2006; Tsamardinos *et al.* 2006; Perrier *et al.* 2008).

In this work the algorithms performance is measured by comparing the ability of OPC and PC algorithm to reproduce, from simulated data, a given graph. In detail, since the main difference between PC and OPC algorithm is in the skeleton identification phase, three indexes based on underlying graphs comparison are computed. These measures, discussed in Section 4.1.1, compare the skeleton of the true DAG with the skeleton of the estimated DAG in term of sensitivity, specificity and precision. The structural accuracy of algorithms, instead, is measured by a performance indicator, introduced in Section 4.1.2 that computes the structural distance between PDAGs.

4.1.1 True positive rate, false positive rate and true discovery rate

In this Section we deals with some performance indicators used to evaluate the ability of identifying the true skeleton.
Let G^* be the skeletons of the true graph and H^* be the skeleton of the estimated graph. The measures are:

- *True positive rate - TPR.* This index is given by the number of edges correctly found in H^* over the number of true edges in G^*.

$$TPR = \frac{\text{edges correctly found in } H^*}{\text{true edges in } G^*} \tag{4.1}$$

  The TPR is equal to 1 when H^* matches G^* and it is 0 when the true graph contains no edges.
  Generally speaking, the TPR measures the proportion of actual positives which are correctly identified as such. In some fields, the TPR is called *sensitivity* or *recall* rate that is a measure of completeness.

- *False positive rate - FPR.* This index is given by the number of edges incorrectly found in H^* over the number of true gaps (absent edges) in G^*.

$$FPR = \frac{\text{edges incorrectly found in } H^*}{\text{true gaps in } G^*} \tag{4.2}$$

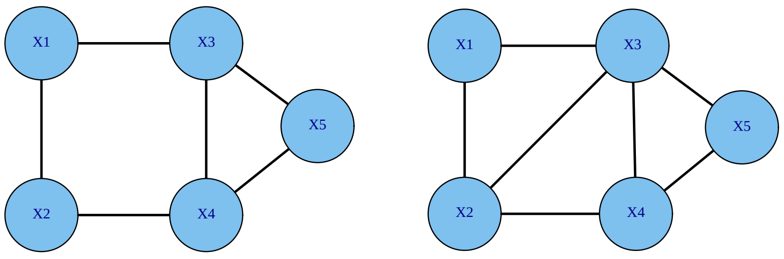
The FPR is equal to 0 when H^* matches G^* and it is 1 when the true graph contains no gaps.
The FPR measures the proportion of negatives which are correctly identified; in our case it is the proportion of edges erroneously found. The FPR is also known as the complement to one of *specificity*.

- *True discovery rate - TDR*. This index is the proportion of edges correctly found on the total number of found edges (both in the estimated graph).

$$TDR = \frac{\text{edges correctly found im } H^*}{\text{found edges in } H^*} \tag{4.3}$$

The TDR is supposed to be equal to 1 when edges are all correctly found in the estimated graph. The TDR is conventionally equal to 1 when there are no edges in both true and estimated graph. If the true graph is empty while the estimated one is not, TDR is zero. It coincides with the positive prediction value that is a measure of exactness so, the TDR is also known as *precision*.

An example of TPR, FPR and TDR is reported in the following. Let the



(a) G^* i.e. the skeleton of the true DAG.

(b) H^* i.e. the skeleton of the estimated DAG.

Figure 4.1: Graph skeletons

graph in Figure 4.1(a) be the skeleton G^* of the true DAG and let the graph in Figure 4.1(b) be the skeleton H^* of the estimated DAG. G^* has six edges and H^* has seven edges; six out of the seven edges in H^* are correct but there is an extra edge between X_2 and X_3. In addition, G^* has four gaps. For this example the TPR, the FPR and the TDR are:

- $TPR = \frac{6}{6} = 1$
- $FPR = \frac{1}{4} = 0.25$
- $TDR = \frac{6}{7} = 0.86$

In this example, all measures are quite satisfying. The TPR that represents the sensitivity, achieves the maximum; the FPR that represents (1-specificity) is quite small; the TDR, standing for precision, is high.
It is worth noting that the TPR and the FPR are closely related to the type I and type II errors. In detail, the TPR is the complement to one of the false negative rate that corresponds to the probability β of type II error; the FPR corresponds to the probability α of type I error. When the sensitivity is high, then β is low and the power of the test is high; when the FPR is low, α is low as well.
These measures are largely discussed in the literature; for further details see Baldi *et al.* (2000) and Dey *et al.* (2009). In `R` it is possible to compute these indexes by means of the *compareGraphs* function of `pcalg` package (Kalisch *et al.* 2009).

4.1.2 Structural Hamming Distance

The *Structural Hamming Distance* - SHD (Tsamardinos *et al.* 2006) - is a performance measure that computes the structural distance between PDAGs. A PDAG is graph containing both directed and directed edges and that represent an equivalence class (see Section 2.3 for further details). SHD is an overall metric that directly compares the learnt PDAG and the original PDAG by counting the number of operations required to convert the fitted graph into the true graph. Admitted operations are addition or deletion of edges and addition, deletion or reversal of directions. In detail, SHD is the sum of five components:

- extra edge error (EE): number of edges learnt in the estimated graph that do not exist in the true one;

- missing edge error (ME): number of edges missing in the estimated graph that are in the true one;
- extra direction error (ED): number of edge directions that appear in the estimated graph but not in the true one;
- missing direction error (MD): number of edge directions that appear in the true graph but not in the estimated one;
- reversed direction error (RD): number of edge directions in the estimated graph that are opposite in the true one;

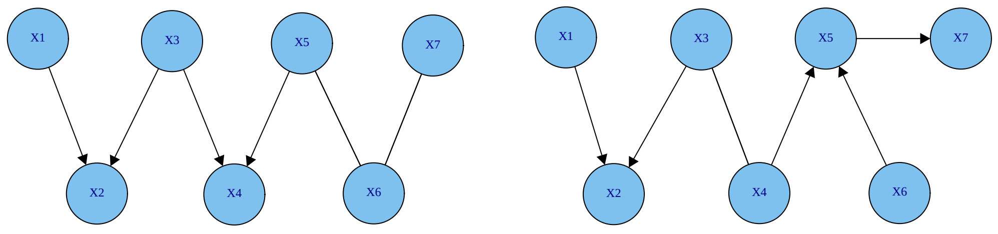
The last three metrics - ED, MD and RD - can be summarised in the directional error (DE) that counts the number of edges incorrectly oriented in the learnt graph. The score, thus, is a raw sum of EE, ME and DE; the score is increased by 1 point for each required operation. Small values of SHD suggest that H is close to G.
A pseudo-code of the procedure is shown in Algorithm 4 where, following the notation of Tsamardinos *et al.* (2006), the learnt PDAG is denoted by H and the true PDAG by G.

Algorithm 4 SHD (Learnt PDAG H versus True PDAG G)

```
$shd = 0$
for every edge $E$ different in $H$ than $G$ do
  if $E$ is missing in $H$ then
    $shd = shd + 1$
  end if
  if $E$ is extra in $H$ then
    $shd = shd + 1$
  end if
  if $E$ is incorrectly oriented in $H$ then
    $shd = shd + 1$
  end if
end for
```

In the following we propose an example useful to describe the SHD computation.

Let the graph in Figure 4.2(a) be the PDAG of the true DAG and let the graph in Figure 4.2(b) be the PDAG of the estimated DAG.



(a) The PDAG of G (b) The PDAG of H

Figure 4.2: Partially DAGs

The SHD value is the sum of operations required to make PDAGs match. More specifically, comparing H with G we can observe that:

- there is an extra edge between X_5 and X_7 (this is the EE component);
- there is a missing edge between X_6 and X_7 (this is the ME component);
- there is a missing direction in the edge between X_3 and X_4 (this is the MD component);
- there is a reverse direction in the edge between X_4 and X_5 (this is the RD component);
- there is an extra direction between X_6 and X_5 (this is the ED component).

We require five operations to transform H into G. Since each operation increases 1 point the score, the SHD is equal to 5.
The reason for computing the SHD score on PDAGs rather than DAGs is that DAGs are not statistically distinguishable. So, the SHD score should be subject to another penalization if computed on DAGs. If only DAGs are available, before calculating the score, it is necessary to convert DAGs to the corresponding PDAGs using the algorithm in Chickering (1995). Generally speaking, the Chickering algorithm converts a DAG in a PDAG through

a *local transformation* of the DAG structure. The procedure consists in two main steps: firstly a total ordering over edges is defined according to the nodes ordering; then compelled edges are found. Since compelled edges encode (conditional) independence statements that occur in all structures of the class, the identification of compelled edges is the crucial point of the Chickering algorithm .
In `R` it is possible computing the SHD metric by means of the *shd* function of `pcalg` package (Kalisch *et al.* 2009).
In the following Section some empirical evaluations are presented.

4.2 Empirical evaluation

The aim of the experimental evaluation is to compare the PC and OPC performance. Two simulation studies have been carried out. We used two sets of ordinal data: the first one comes from a real customer satisfaction survey, and the other deals with a survey well known in literature (Barnes and Kaase 1979). For each dataset, a Bayesian network has been learnt using the PC algorithm; the learnt network has been assumed as true and used as gold standard. According to every network, we generated multiple datasets for different sample sizes: 50, 100 and 500. In detail, we generated 1000 training sets for each sample size. For each simulated dataset two Bayesian networks have been estimated: one using the PC algorithm and one using the OPC algorithm, given a level of significance equal to 0.05. We compared results of structural learning with the gold standard in term of (1) skeleton identification ability and (2) structural accuracy. We computed a mean value of each performance measure with respect to different sample sizes. We present results of simulation studies for each experimental evaluation.

4.2.1 Costumer satisfaction data

The first experiment is conducted using data coming from a real customer satisfaction survey. The survey has been carried out in 2008 with the goal of measuring the citizens satisfaction about services delivered by an Italian post office. Data have been collected by a questionnaire consisting of several items. Respondents answered the sentences using an anchored seven integer scale going from 1 (I strongly disagree) to 7 (I strongly agree), so measured variables are ordinal. The questionnaire has been administered to 228 users. We considered a selection of six items of the questionnaire. Due to the sparsity of contingency tables, variables categories have been merged in three levels. Selected variables refer to the following concepts:

- *Reliability*: capability to deliver the service in a reliable and accurate way;
- *Reassurance capacity*: employers competence and courtesy and their capability to ensure customers;
- *Tangible aspect*: physical building and facilities aspect,
- *Empathy*: customer care and service customization;
- *Answer ability*: willing in helping customers and capability to deliver the service quickly;
- *Overall satisfaction*: customers' perception about global satisfaction.

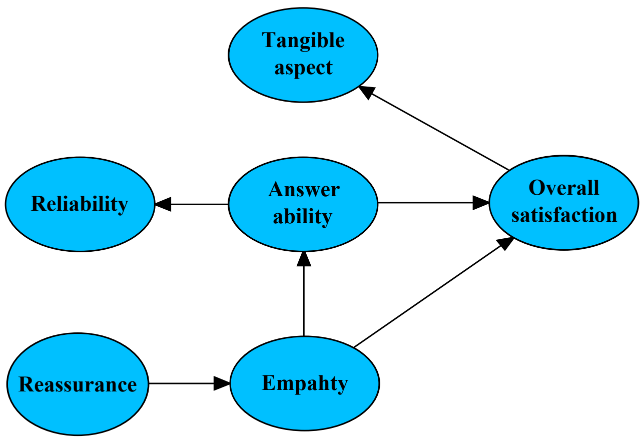


Figure 4.3: The true DAG for customer satisfaction data

The literature discusses many methods for analysing customer satisfaction data (Grigoroudis and Siskos 1998; Zanella 2001; Hayes 1998; Cassel 2000) and some applications of graphical models have been proposed (Salini and Kenett 2007; Musella *et al.* 2007; Figini 2010; Renzi *et al.* 2009). Our goal is to measure the OPC algorithm performance, so data have been used to learn a network that has been considered as the gold standard. The DAG is reported in Figure 4.3.
According to this structure 1000 datasets have been generated for each of these different sample sizes: 50, 100 and 500. PC and OPC algorithm have been performed and performance measures have been computed on each learnt structure. The following tables show the average, over 1000 runs, of the TPR, FPR and the TDR respectively for both algorithms.

	Sample size		
Algorithm	**50**	**100**	**500**
PC	0.42	0.64	0.97
OPC	0.69	0.87	0.97

Table 4.1: The TPR average for PC and OPC algorithm performed on 1000 datasets generated according to the DAG in Figure 4.3

	Sample size		
Algorithm	**50**	**100**	**500**
PC	0.11	0.04	0.02
OPC	0.04	0.03	0.02

Table 4.2: The FPR average for PC and OPC algorithm performed on 1000 datasets generated according to the DAG in Figure 4.3

We can observe that the skeleton learning ability of both algorithms increases with larger sample sizes. In detail since more observations makes statistical test more sensitive, when the sample size is equal to 500, the TPR is really close to one for both algorithms. Furthermore, for the largest sample size, algorithms have the same performance results considering every indexes.

	Sample size		
Algorithm	**50**	**100**	**500**
PC	0.73	0.91	0.98
OPC	0.93	0.96	0.99

Table 4.3: The TDR average for PC and OPC algorithm performed on 1000 datasets generated according to the DAG in Figure 4.3

Despite that, when the sample size is 50 or 100 the TPR and the TDR of OPC algorithm are larger than those of PC algorithm and the FPR of OPC algorithm is smaller than that of PC algorithm. The gap of performance is significant for the smallest sample size. On the basis of this results, the OPC algorithm seems to be more sensitive, more accurate and more reliable than PC algorithm above all for small samples.
Another performance measure that has been computed is the SHD. The Table 4.4 shows the average over 1000 runs of the SHD for both algorithms.

	Sample size		
Algorithm	**50**	**100**	**500**
PC	6.07	4.92	3.3
OPC	4.89	4.17	3.2

Table 4.4: The SHD average for PC and OPC algorithm performed on 1000 datasets generated according to the DAG in Figure 4.3

The SHD of both algorithms decreases when the sample size increases. However, given the same sample size, the SHD of PC algorithm is larger than the SHD of OPC algorithm. Since small values of SHD mean that less operations are required to make the estimated PDAG and the true PDAG match, the OPC algorithm outperforms the PC algorithm.

Figure 4.4 shows the behaviour of SHD for both algorithms.

Whit respect to each sample size, the SHD of OPC algorithm is smaller than SHD of PC algorithm. This means that, according to this experiment,

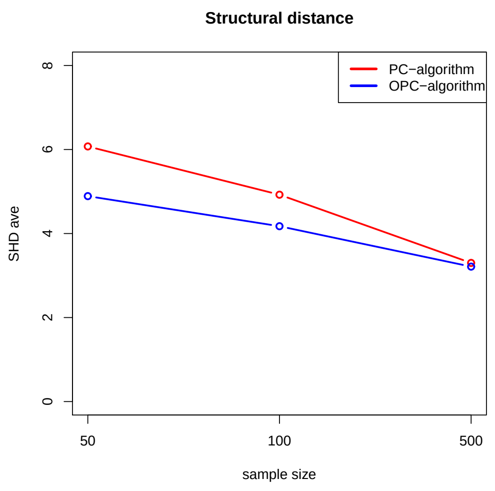


Figure 4.4: The SHD average for PC algorithm (red line) and OPC algorithm (blue line) with respect of different sample sizes for Customer Satisfaction data

the OPC algorithm outperforms the PC algorithm in term of structural distance between PDAGs.

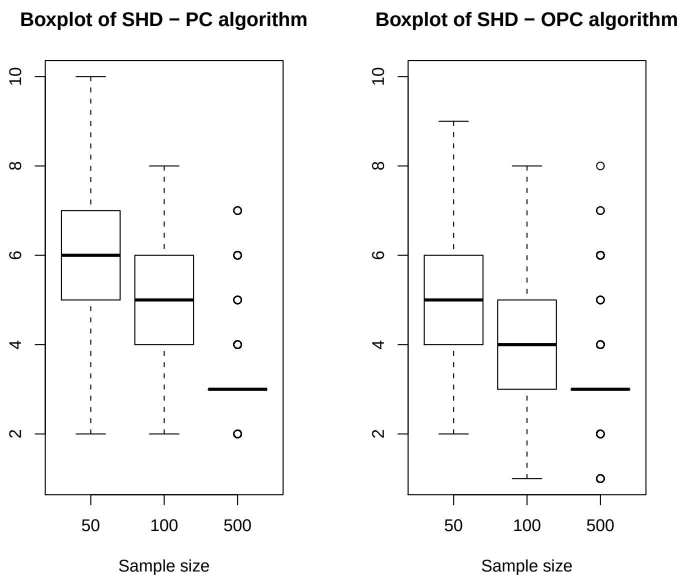


Figure 4.5: Box plots of SHD according to different sample sizes for Customer Satisfaction data.

Figure 4.5 shows box plots of SHD for both algorithms according to different sample sizes.

- PC: when the sample size is 50, the SHD distribution is symmetric and it varies between 2 and 10. When the sample size is 100, the distribution is still symmetric but it is concentrated between smaller values (2 and 8). When the sample size is 500, the SHD almost always assumes value 3.
- OPC: SHD distribution varies between 2 and 9 when the sample size is 50 and between 1 and 8 when the sample size is 100. When the sample size is 500, the distribution is concentrated on 3. In this case there are several outliers.

The Figure shows that the median value of SHD computed for OPC algorithm is smaller than that computed for PC algorithm, when the sample size is 50 or 100. For the largest sample, the behaviour of algorithms is similar. Table 4.5 gives a summary of some statistics relating to SHDs.

Algorithm	**Statistics**	**Sample size**		
		50	**100**	**500**
PC	*Lower whisker*	2.00	2.00	3.00
	Q_1	5.00	4.00	3.00
	Q_2	6.00	5.00	3.00
	Q_3	7.00	6.00	3.00
	Upper whisker	10.00	8.00	3.00
	C_v	26.52%	23.81%	30.12%
OPC	*Lower whisker*	2.00	1.00	3.00
	Q_1	4.00	3.00	3.00
	Q_2	5.00	4.50	3.00
	Q_3	6.00	5.00	3.00
	Upper whisker	9.00	8.00	3.00
	C_v	28.33%	33.67%	31.95%

Table 4.5: Statistics relating to SHDs of Customer Satisfaction data

It is worth noting that the SHD distribution of OPC algorithm is more variable (in term of coefficient of variation) than the SHD distribution of PC algorithm in each case (n = 50, 100, 500). In order to verify the reliability of these results for different data, another simulation study has been conducted: the results are presented in the following Section.

4.2.2 Political Action data

In this Section, we present another experimental study. Data used come from a cross-national survey conducted by Barnes and Kaase (1979) with the aim to gather information about political participation in industrial societies. This survey, known in the literature as Political Action survey, was carried out on a sample of persons 16 years and older. The Political Action Survey collected data about attitude and opinions and contains several hundred variables. However, only some variables have been used. Selected variables refer to the definition of political efficacy that is "the feeling that individual political action does have, or can have, an impact upon the political process" (Campbell *et al.* 1954). Data taken into account in this work refer to the opinions of a set of USA respondents about to the following six sentences of the questionnaire:

- *NoSay*: people like me have no say in what the government does;
- *Voting*: voting is the only way that people like me can have any say about how the government runs things;
- *Complex*: sometimes politics and government seem so complicated that a person like me cannot really understand what is going on;
- *NoCare*: I dont think that public officials care much about what people like me think;
- *Touch*: generally speaking, those we elect to Congress in Washington lose touch with people pretty quickly;
- *Interest*: parties are only interested in peoples votes but not in their opinions.

Respondents answered the questions using a semi-ordinal scale with five levels: *strongly agree, agree, disagree, strongly disagree, don't know*. The responses *don't know* can not be considered as a category in an ordinal scale and they are usually handle as missing values. However, in order to have a complete dataset with ordinal variables, only full cases have been taken into account. Dataset used for learning the DAG assumed as true is thus made by six ordinal variables with four levels measured on 768 units.
The DAG of this data is displayed in Figure 4.6

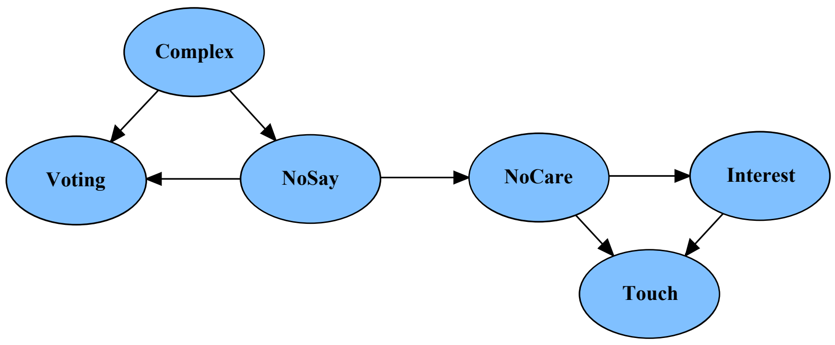


Figure 4.6: The true DAG for Political Action data

The DAG in Figure 4.6 represents the structure from which a training set of 1000 datasets have been generated using different sample sizes (i.e 50, 100 and 500). Both PC and OPC algorithm have been performed on each dataset. The TPR, FPR and TDR average on 1000 runs have been computed. The achieved results are presented in the following tables.

	Sample size		
Algorithm	**50**	**100**	**500**
PC	0.38	0.50	0.88
OPC	0.61	0.79	0.95

Table 4.6: The TPR average for PC and OPC algorithm performed on 1000 datasets generated according to the DAG in Figure 4.6

	Sample size		
Algorithm	**50**	**100**	**500**
PC	0.11	0.07	0.00
OPC	0.02	0.01	0.00

Table 4.7: The FPR average for PC and OPC algorithm performed on 1000 datasets generated according to the DAG in Figure 4.6

	Sample size		
Algorithm	**50**	**100**	**500**
PC	0.76	0.86	0.99
OPC	0.96	0.98	0.99

Table 4.8: The TDR average for PC and OPC algorithm performed on 1000 datasets generated according to the DAG in Figure 4.6

Results seem suggest the same consideration as in the previous experiment. Algorithms performance increase with the increasing of sample size. In detail, OPC algorithm performance are more satisfying that that of PC algorithm when the sample size is 50 or 100. When the sample size is 500 the behaviour of PC and OPC is almost equal.
The SHD values computed in this experiment are presented in Table 4.9.

	Sample size		
Algorithm	**50**	**100**	**500**
PC	7.22	6.73	5.41
OPC	6.56	6.38	4.50

Table 4.9: The SHD average for PC and OPC algorithm performed on 1000 datasets generated according to the DAG in Figure 4.6

The SHD values decrease when the sample size increase. The SHD values are smaller if referred to OPC algorithm than to PC algorithm. Figure 4.7 shows the trend over the sample size.

Both algorithms significantly increase their capability to learn the PDAG when the sample size is 500. Figure 4.8 shows box plots of SHD for both al-

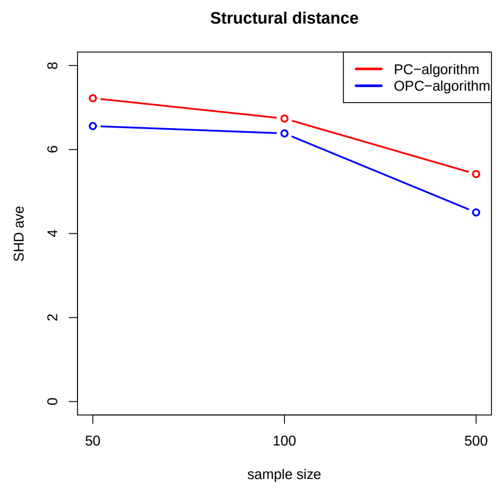


Figure 4.7: The SHD average for PC algorithm (red line) and OPC algorithm (blue line) according to different sample sizes.

gorithms across the different sample sizes. The Figure suggests the following consideration:

- PC: both range and midspread reduces when the sample size increases. When the sample size is 50, the distribution varies between 10 and 5 and has a positive symmetry. When the sample size is 100, the distribution is more symmetric and it varies between 4 and 9. When the sample size is 500 the range reduces into 3 and the distribution has a positive symmetry.
- OPC: When the sample size is 50, the third quartile coincides with the median. This suggests that the distribution has a negative asymmetry. When the sample size is 100, the first quartile coincides with the median. In this case, the distribution has a positive asymmetry. This holds also when the sample size is 500.

The SHD distribution of PC algorithm is more variable (in term of coefficient of variation) than that of OPC algorithm when the sample size is 50 or 100. When the sample size is 500, the SHD distribution of PC is less variable. Table 4.10 gives a summary of some statistics relating to SHDs.

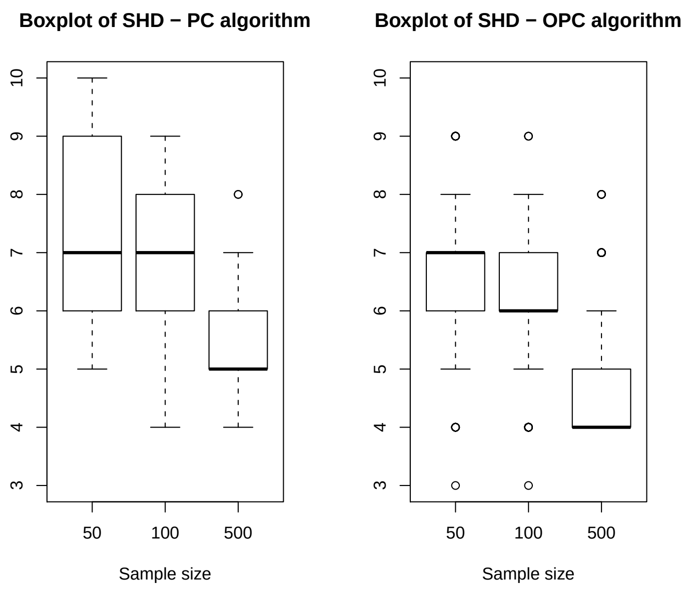


Figure 4.8: Box plots of SHD according to different sample sizes for Political Action data.

Algorithm	**Statistics**	**Sample size**		
		50	**100**	**500**
PC	*Lower whisker*	5.00	4.00	4.00
	Q_1	6.00	6.00	5.00
	Q_2	7.00	7.00	5.00
	Q_3	9.00	8.00	6.00
	Upper whisker	10.00	9.00	7.00
	C_v	18.75%	20.14%	14.83%
OPC	*Lower whisker*	5.00	5.00	4.00
	Q_1	6.00	5.00	4.00
	Q_2	7.00	6.00	4.00
	Q_3	7.00	7.00	5.00
	Upper whisker	8.00	8.00	6.00
	C_v	13.34%	13.52%	19.47%

Table 4.10: Statistics relating to SHDs of Political Action data

4.3 Further considerations

Results of simulation studies presented in Sections 4.2.1 and 4.2.2 suggest that OPC algorithm outperforms the PC algorithm in presence of ordinal

data and when the sample size is small. However, these results verifies only if a monotone association between variables exists. If variables are dependent but not monotonically associated, the test used for checking conditional independences in the OPC algorithm, the Jonckheere-Terpstra, fails. An example can be useful for showing the OPC behaviour when the association between variables is not monotonic.
Consider the DAG in Figure 4.9. The graph represents the conditional independence relations among three ordinal variables. According to the graph it is possible saying that $X_1 \perp\!\!\!\perp X_3$ given X_2.

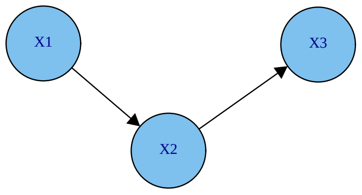


Figure 4.9: A DAG on three variables non monotonically associated

Let $\boldsymbol{X}$=(X_1, X_2, X_3) be the set of variables belonging to the dataset from which the DAG in Figure 4.9 has been learnt using PC algortihm. As discussed in Section 3.1.2, the PC algorithm consists in three main steps: (1) starting with a complete undirected graph the DAG skeleton is found by checking independences; (2) the colliders are identified; (3) the rest of the nodes are oriented without producing any cycle or other colliders. These steps are the same for the OPC algorithm. The relevant difference between PC and OPC algorithm is in the statistical test used in the first step: the PC algorithm computes the mutual information between variables and it takes decisions according to the value of G^2 statistical test; the OPC algorithm according to the results of the Jonckheere-Terpstra test. Despite that, both algorithms proceed checking independences between pair of nodes giving a subset of increasing cardinality.
In this example, we reproduce the output of the first step of both algorithms in order to show the different behaviour of algorithms when variables are non monotonically associated. The starting point of both algorithms is the

complete undirected graph in Figure 4.10. The significance level used is 0.05.

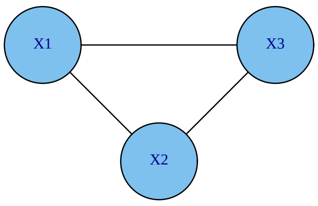


Figure 4.10: The complete undirected graph associated with the DAG in Figure 4.9

We start setting the cardinality of the separator set equal to zero and testing marginal independence between the following ordered pairs of variables:

- (X_1, X_2);
- (X_1, X_3);
- (X_2, X_3).

The first independence test is performed on the pair (X_1, X_2) with respect to the contingency table in Table 4.11.

	X2			
X1	**1**	**2**	**3**	**Sum**
1	69.00	66.00	32.00	**167.00**
2	23.00	80.00	61.00	**164.00**
3	72.00	53.00	44.00	**169.00**
Sum	**164.00**	**199.00**	**137.00**	**500.00**

Table 4.11: Contingency table for variables X_1 and X_2

Analysing the contingency table, it is possible noting that X_1 and X_2 are weakly associated. This is suggested by the *Cramer* index V (Cramér 1946) that is a relative measure of association between two variables. V varies between 0 (reflecting complete independence) and 1 (reflecting complete

dependence or association) and it is equal to 0.20 in this case. As discussed in Section 3.4, in presence of ordinal variables a monotone association is common so it could be interesting to verify if the association between X_1 and X_2 is monotone. An useful measure of monotone association is *Gamma*. Close to zero it shows very little monotone association between variables. In our example, γ is 0.03 that seems to be small enough to say there is not a monotone association between X_1 and X_2. So, variables are dependent but non monotonically associated. For these reasons, the test based on the conditional cross entropy, on which the PC algorithm is based, and the Jonckheere-Terpstra test, used in the OPC algorithm, give different results. Performing the PC algorithm the p-value associated with independence test between X_1 and X_2 is really close to zero: this suggests to reject the null hypothesis of independence and the edge between X_1 and X_2 is kept in the complete starting graph. On the contrary, performing the OPC algorithm, the p-value is 0.5145 that suggests there is no evidence in data to reject the null hypothesis and the edge between X_1 and X_2 is deleted from the complete starting graph.
Consider the second pair of variables. The contingency table for (X_1, X_3) is shown in the Table 4.12.

Again there is a weak association between variables ($V = 0.11$) and an

	X3			
X1	**1**	**2**	**3**	**Sum**
1	66.00	59.00	42.00	**167.00**
2	80.00	58.00	26.00	**164.00**
3	53.00	75.00	41.00	**169.00**
Sum	**199.00**	**192.00**	**109.00**	**500.00**

Table 4.12: Contingency table for variables X_1 and X_3

unimportant monotone association ($\gamma = 0.06$). The p-value obtained using PC algorithm is equal to 0.011 while its value is 0.3014 when the OPC algorithm is performed. Algorithms decide in a different way also in this case. The edge between X_1 and X_3 is kept if the PC algorithm is performed, is deleted using the OPC algorithm.

Consider the third pair of variables. The contingency table for the pair (X_2, X_3) is the Table 4.13.

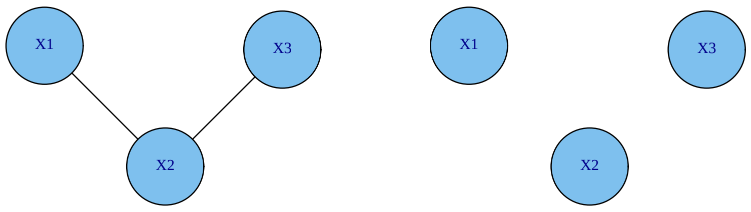
	X3			
X2	**1**	**2**	**3**	**Sum**
1	0.00	105.00	59.00	**164.00**
2	199.00	0.00	0.00	**199.00**
3	0.00	87.00	50.00	**137.00**
Sum	**199.00**	**192.00**	**109.00**	**500.00**

Table 4.13: Contingency table for variables X_2 and X_3

Variables are associated ($V = 0.70$) but the value of γ is quite small again ($\gamma = -0.07$). For this pair of variable the PC algorithm decides to keep the edge since the p-value associated with the independence test based on conditional cross entropy between X_2 and X_3 is close to zero. Since the p-value of Jonckheere-Terpstra test is equal to 0.10, the OPC algorithm deletes the edge between X_2 and X_3.
At this point of the procedure, the cardinality of separator set is increased by one. However, the OPC algorithm stops since, on the basis of the taken decisions, there is no node with a number of adjacent nodes bigger than the new cardinality of separator set (i.e. one). On the contrary, PC algorithm checks the following conditional independences:

- $X_1 \perp\!\!\!\perp X_2 | X_3$: the test gives a p-value=0.000 that suggests to keep edge between X_1 and X_2;
- $X_1 \perp\!\!\!\perp X_3 | X_2$: the test gives a p-value=0.988 that strongly suggests to remove edge between X_1 and X_3;
- $X_2 \perp\!\!\!\perp X_3 | X_1$: the test gives a p-value close to zero that suggests to keep edge between X_2 and X_3.

After having checked those conditional independences, the PC algorithm stops since there is no node with a number of adjacent nodes greater than or equal to the new cardinality of the separator set. Skeletons identified by algorithms are displayed in Figure 4.11.



(a) The skeleton estimated through PC algorithm

(b) The skeleton estimated through OPC algorithm

Figure 4.11: Skeletons identified by algorithms

According to the graph in Figure 4.11(b) variables are all independent to each other since no edge has been found by OPC algorithm. The PC algorithm, instead, identifies the skeleton in Figure 4.11(a) where there are two edges. Different skeletons are due to the behaviour of algorithms. Decisions taken at each step of the OPC algorithm have been completely affected by the absence of monotone association between variables. When variables are ordinal but there is not a monotone association, the OPC algorithm is not reliable and the PC algorithm has to be used for learning the dependence structure.

Conclusions

Summary of the main results

This thesis dealt with the problem of learning a Bayesian network when subject-matter knowledge is not available. In this situation it is necessary to infer the network from data using automatic procedures for learning the relations among variables. The main purpose of this dissertation was to develop a new procedure for Bayesian networks structural learning in presence of ordinal variables. A constraint-based approach has been adopted. The motivation of the work is basically due to the fact that existent learning algorithms do consider ordinal variables as nominal. This determines a loss of a part of that information inherent to ordering among categories of ordinal variables. The aim of the work has been achieved by proposing a variation of the PC algorithm namely OPC algorithm. More specifically, algorithm structures are the same, they differ in the test used for checking conditional independence. The OPC statistical procedure is, indeed, based on a nonparametric rank-based test appropriate for ordinal variables and called Jonckheere-Terpstra test. As a consequence of this, the OPC algorithm represents an opportunity to learn the network without demoting ordinal variables in nominal. Its performance has been compared with that of PC algorithm in term of sensitivity (computing the TPR), specificity (by the FPR) and precision (using the FPR) as well as structural accuracy (by means of SHD). In detail, TPR, FPR and TDR have been computed in order to compare the ability of PC and OPC algorithms to estimate the DAG skeleton; the SHD has been used to verify their capacity to draw the true DAG. Empirical evaluations have been carried out for two different sets of data: Customer Satisfaction data and Political data. Main results, obtained by taking the mean over the 1000 replications, are encouraging and are briefly summurised in the following:

1. *Customer Satisfaction data:* for small sample size (50 or 100) the OPC algorithm is more precise in the skeleton identification and more accurate in DAG estimation; for larger sample sizes (500), there is any difference in algorithm behaviours.

2. *Political data:* for small sample size (50 or 100) the OPC algorithm is more accurate than PC algorithm both in the skeleton identification and in DAG estimation; for larger sample sizes (500) algorithms behave similarly.

According to the empirical evaluation proposed in this work, in presence of ordinal variables and restricted sample size, the OPC algorithm represents a more suitable solution for the structural learning matter. However, the new procedure shows some limitations that are discussed in the following Section.

Open problems

The OPC algorithm is an attempt to overcome the PC algorithm limitation of treating all categorical variables as nominal. However, the PC algorithm has some other relevant limitations. These, discussed and treated by different authors (see Section 3.1.3 more details), have determined the development of both alternative versions and variations of PC algorithm. Nevertheless, the OPC algorithm evolves from the original version of PC algorithm so it inherits all its growing limitations. In addition, the OPC algorithm has further limitations concerning both the statistical procedure and the computational aspect. In particular:

1. OPC algorithm is suitable when all variables are ordinal; realistically, in observational studies some features are measured on a nominal scale and other on an ordinal scale. Therefore datasets contain both nominal and ordinal variables. The OPC algorithm is not appropriate for mixed variables and it is necessary to resort to the PC algorithm treating all variables as nominal.

2. The Jonckheere-Terpstra test only checks for monotone association between ordinal variables. When ordinal variables are associated but not monotonically, the test fails. In addition, the Jonckheere-Terpstra test requires that alternative hypothesis is arranged in a specific order. This entails that ordering should be specified before the data are collected.

3. We did not treat the problem of sparse tables. Sparseness is common in tables with very variables or when the sample size is small. In presence of sampling zeros, some computational problems can verify. For instance, p-value can not be available.

These considerations highlight some open problems and lead to some natural extensions that can become objects of further research.

1. In order to respect all different types of categorical variables, a novel algorithm, called NOPC, has been proposed in this dissertation. The NOPC algorithm represents a natural development of OPC algorithm and it is suitable to learn a network from mixed nominal and ordinal data. The statistical procedure is based on a set of nonparametric rank-based tests checking conditional independence between pairs of categorical variables; in detail, after having settled the class of each variable in the dataset (nominal, ordinal or binary), the procedure automatically selects, at each iteration, the most appropriate test according to the pair of variables involved. The OPC algorithm procedure is discussed in Section 3.4 and the `R` code is available in Appendix A. However, we did not test the performance of NOPC algorithm. This aspect can be the object of further research.

2. There are some alternatives of computing tests. In presence of sparse tables, an efficient way to estimate the p-value observed is by using *sequential Monte Carlo sampling*. A future research could deal with the algorithm improvement in presence of sparse tables.

Bibliography

Abellán, J., Gómez-Olmedo, M., and S., S. M. (2006). Some variations on the PC Algorithm. In *Proceedings of Probabilistic Graphical Models*, pp. 1–8.

Agresti, A. (2002). *Categorical Data Analysis.* Wiley-Interscience, New Jersey.

Agresti, A. (2010). *Analysis of ordinal data.* Wiley, New Jersey.

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

Baldi, P., Brunak, S., Chauvin, Y., Andersen, C., and Nielsen, H. (2000). Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, **16**, (5), 412–9.

Barnes, S. H. and Kaase, M. (1979). *Political Action: Mass Participation in Five Western Democracies.* Sage Pubblications, Beverly Hills, California.

Berge, C. (1973). *Graph and Hypergraph.* North-Holland, Ansterdam.

Blalock, H. M. J. (1971). *Causal models in the social sciences.* Aldine-Atheston, Chicago.

Blanco, R., Inza, I., and Larraaga, P. (2003). Learning Bayesian networks in the space of structures by estimation of distribution algorithms. *International Journal of Intelligent Systems*, **18**, 205–20.

Buntine, W. (1994). Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, **2**, 159–225.

Buntine, W. (1996). A guide to the literature on learning probabilistic networks from data. *IEEE Transaction on Knowledge and Data Engineering*, **8**, 195–210.

Campbell, A., Gurin, G., and Miller, W. E. (1954). *The Voter Decides.* Row and Peterson, Evanston, IL.

Cano, A., Gómez-Olmedo, M., and Moral, S. (2008). A Score Based Ranking of the Edges for the PC Algorithm. In *Proceedings of the Fourth European Workshop on Probabilistic Graphical Models*, pp. 41–8.

Cassel, C. M. (2000). Measuring customer satisfaction on a national level using a superpopulation approach. *Total quality management*, **11:7**, 909–15.

Chickering, D. M. (1995). A transformational characterization of Bayesian network structures. In *Proceedings of the Eleventh Conference on Uncertainty in Artifcial Intelligence (UAI-95)*, pp. 87–98. Morgan Kaufmann, San Francisco. CA.

Chickering, D. M. (2002). Learning Equivalence Classes of Bayesian-Network Structures. *Journal of Machine Learning Research*, **2**, 445–498.

Chickering, D. M., Heckerman, D., Meek, C., and Madigan, D. (1994). Learning Bayesian networks is NP-hard. Technical Report MSR-TR-94-17, Microsoft Research, Microsoft Corporation, Redmond, Washington.

Clogg, C. C. and Shihadeh, E. S. (1994). *Statistical Models for Ordinal Variables.* Sage Pubblications, Thousand Oaks, California.

Cooper, G. and Herskovits, E. (1992). A Bayesian method for constructing Bayesian belief networks from databases. *Machine Learning*, **9:(4)**, 309–47.

Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory.* John Wiley and Sons, New Jersey.

Cowell, R. G., Dawid, A. P., Lauritzen, S. L., and Spiegelhalter, D. J. (1999). *Probabilistic Networks and Expert Systems.* Springer, New York.

Cox, D. and Wermuth, N. (1996). *Multivariate Dependencies. Models, analysis and interpretation.* Chapman and Hall/CRC, Boca Raton, Florida.

Cox, D. and Wermuth, N. (2004). Joint response graphs and separation induced by triangular systems. *Journal of the Royal Statistics Society, B*, **66**, 687–717.

Cramér, H. (1946). *Mathematical methods of statistics*. Princenton, Uppsola, Swedan.

Cruz-Ramírez, N., Acosta-Mesa, H.-G., Barrientos-Martínez, R.-E., and Nava-Fernández, L.-A. (2006). How Good Are the Bayesian Information Criterion and the Minimum Description Length Principle for Model Selection? A Bayesian Network Analysis. In *MICAI*, pp. 495–504.

Csárdi, G. and Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, **Complex Systems**, 1695.

Dawid, A. P. (1979). Conditional independence in statistical theory (with discussion). *Journal of the Royal Statistical Society, Series B*, **41**, 1–31.

Dawid, A. P. (1980). Conditional independence for statistical operations. *Annals of Statistics*, **8**, 598–617.

de Campos, L. M., Fernandez-Luna, J. M., Gomez, J. A., and Puerta, J. M. (2002). Ant colony optimization for learning Bayesian networks. *International Journal of Approximate Reasoning*, **31**, 511–49.

Dey, D. K., Ghosh, S., and Mallick, B. K. (2009). *Bayesian Modeling in Bioinformatics*. Chapman and Hall/CRC, Boca Raton, Florida.

Edwards, D. (1995). *Introduction to Graphical Modelling*. Springer, New York.

Fernandes, C. M., Silva, W. T. D., and Ladeira, M. (2004). An algorithm to handle structural uncertainties in learning Bayesian network. In *Proceedings of Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC'04)*. Jornadas Iberoamericanas de Ingenieria de Software e Ingenieria del Conocimiento (JIISIC), Madrid, Espana. Jornadas Iberoamericanas de Ingenieria de Software e Ingenieria del Conocimiento (JIISIC).

Figini, S. (2010). Statistical models for customer satisfaction data. *Journal of quality technology and quality management - to appear*, **7**.

Geiger, D. and Pearl, J. (1990). On the logic of casual models. In *Uncertainty in artificial intelligence IV*, pp. 136–47. (ed. R.D. Schacter, T.S. Levitt, L.N. Kanal and J.F. Lemmer), North-Holland, Amsterdam.

Geiger, D. and Pearl, J. (1993). Logical and algorithmic properties of conditional independence and graphical models. *Annals of Statistics*, **21**, 2001–21.

Gibbs, W. (1902). *Elementary Principles of Statistical Mechanics.* Yale University Press, NewHaven, Connecticut.

Goodman, L. A. and Kruskal, W. (1979). *Measures of association for cross classification.* Springer-Verlag, New York.

Grigoroudis, E. and Siskos, Y. (1998). *Customer Satisfaction Evaluation. Methods for measuring and implementing services quality.* Springer, Milwaukee, Wisconsin.

Hayes, B. E. (1998). *Measuring Customer Satisfaction: Survey Design, Use, and Statistical Analysis Methods*, (2nd edn). ASQ Quality Press, Milwaukee, Wisconsin.

Heckerman, D. (1995). A tutorial on Learning with Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research.

Hollander, M. and Wolfe, D. (1999). *Nonparametric Statistical Methods*, (2nd edn). John Wiley and Sons, New York.

Joe, H. (1971). *Multivariate Models and Dependence Concepts.* Chapman and Hall/CRC, Boca Raton, Florida.

Jonckheere, A. (1954). A distribution-free k-sample test against ordered alternatives. *Biometrika*, **41**, 133–45.

Kalisch, M. and Bühlmann, P. (2007). Estimating high-dimensional directed acyclic graphs with the pc-algorithm. *Journal of Machine Learning Research*, **8**, 613–636.

Kalisch, M., Maechler, M., and Colombo, D. (2009). *pcalg: Estimation of CPDAG/PAG and causal inference.* R package version 0.1-9.

Kendall, M. G. (1945). The treatment of ties in rank problems. *Biometrika*, **33**, 239–51.

Kiiveri, H., Speed, T., and Carlin, J. (1984). Recursive casual model. *Journal of the Australian Mathematical Society, Series A*, **36**, 30–52.

Konis, K. and Expert., H. (2010). *RHugin: RHugin*. R package version 0.9-1.

Kruskall, W. H. (1952). A nonparametric test for several sample problem. *AMS*, **23**, 525–40.

Kruskall, W. H. and Wallis, W. A. (1952). Use of ranks in one-criterion variance analysis. *JASA*, **47**, 583–621.

Kullback, S. and Leibler, R. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, **22**, 79–86.

Lauritzen, S. L. (1996). *Graphical Models*. Clarendon press, Oxford.

Lauritzen, S. L., Dawid, A. P., Larsen, B. N., and Leimer, H.-G. (1990). Independence properties of directed markov fields. *Networks*, **20**, 491–505.

Lehmann, E. L. (1975). *Nonparametrics: Statistical Methods Based On Ranks*. Holden-Day, San Francisco.

Li, J. and Wang, Z. J. (2009). Controlling the false discovery rate of the association/causality structure learned with the PC algorithm. *Journal of Machine Learning Research*, **10**, 475–514.

Lindgren, B. W. (1976). *Statistical Theory*, (3rd edn). Macmillan Publishing, New York.

Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *AMS*, **18**, 56–60.

Monti, S. and Cooper, G. F. (1997). Learning Bayesian belief networks with neural network estimators. *Neural Information Processing Systems*, **9**, 579–584.

Moral, S. (2004). An empirical comparison of score measures for independence. In *Proceedings of the Tenth International Conference IPMU 2004*, pp. 1307–14.

Musella, F., Renzi, M. F., and Vicard, P. (2007). Un modello di valutazione della qualitá basato su sistemi esperti probabilistici. In *Valutazione e*

Customer Satisfaction per la Qualitá dei servizi, pp. 213–6. Centro stampa universitá, Roma – Italy.

Neapolitan, R. E. (2003). *Learning Bayesian Networks*. Pearson Prentice Hall, NewHaven, Connecticut.

Pearl, J. (1986). A constraint-propagation approach to probabilistic reasoning. In *Uncertainty in Artificial Intelligence*. (ed.L.N. Kanal and J.F. Lemmer), North Holland, Amsterdam, The Netherlands.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems:networks of plausible inference*. Morgan Kaufmann, San Francisco, CA, USA.

Pearl, J. (2000). *Causality: Models, Reasoning and Inference*. Cambridge University Press, New-Yprk.

Pearl, J. and Paz, A. (1987). Graphoids: a graph based logic for reasoning about relevancy relations. In *Advances in Artificial Intelligence - II*, pp. 357–63. North-Holland, Amsterdam.

Pearl, J. and Verma, T. (1987). The logic of representing dependecies by directed graphs. In *Proceedings of the 6th conference of American Association of Artificial Intelligence*, pp. 374–9. American Association of Artificial Intelligence.

Perrier, E., Imoto, S., and Miyano, S. (2008). Finding optimal bayesian network given a super-structure. *Journal of Machine Learning Research*, **9**, 2251–86.

Pirie, W. (1983). Jonckheere tests for ordered alternatives. *Encyclopaedia of Statistical Sciences*, **4**, 315–8.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Renzi, M. F., Vicard, P., Guglielmetti, R., and Musella, F. (2009). Probabilistic expert systems for managing information to improve services. *The TQM Journal*, **21**, 429–42.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, **14**, 465–471.

Salini, S. and Kenett, R. S. (2007). Bayesian networks of customer satisfaction survey data.

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

Shannon, C. (1948). A mathematical theory of communication. *Bell System Technology Journal*, **27**, 379–423.

Siegel, S. and Castellan, N. J. J. (1988). *Nonparametrics: Statistical Methods Based On Ranks*. McGraw-Hill International Editions, New York.

Speed, T. P. (1979). A note on nearest-neighbour gibbs and markov probabilities. *Sankhya, Series A*, **41**, 184–97.

Spirtes, P., Glymour, C., and Scheines, R. (2000). *Causation, Prediction, and Search*, (2nd edn). MIT Press, Cambridge, Massachusetts.

Sprent, P. and Smeeton, S. N. (2001). *Applied Nonparametric Statistical Methods*. Chapman and Hall/CRC, Boca Raton, Florida.

Stanghellini, E. (1999). The use of graphical models in consumer credit scoring. In *Proceedings of the 52nd International Statistical Institute*, pp. 279–82.

Steck, H. (2001). *Constraint-Based Structural Learning in Bayesian Networks using Finite Data*. PhD thesis, Institut für Informatik der Technischen Universität München.

Steck, H. (2007). On a Non-Local Search Strategy for Learning in Bayesian Networks.

Terpstra, T. J. (1952). The asymptotic normality and consistency of Kendalls̀ test against trend when ties are present in one ranking. *Indagationes Mathematicae*, **14**, 327–333.

Tsamardinos, I., Brown, L. E., and Constantin, F. A. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. In *Proceedings of the 52nd International Statistical Institute*, pp. 31–78. Machine Learning.

Verma, T. and Pearl, J. (1990a). Casual networks: semantic and expressiveness. In *Uncertainty in artificial intelligence IV*, pp. 69–76.

(ed. R.D. Schacter, T.S. Levitt, L.N. Kanal and J.F. Lemmer), North-Holland, Amsterdam.

Verma, T. and Pearl, J. (1990b). Equivalence and synthesis of causal models. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pp. 255–70.

Whittaker, J. (1990). *Graphical Models in Applied Multivariate Statistics.* John Wiley and Sons, Chicester, England.

Wilcoxon, F. (1945). Individual comparison by ranking methods. *Biometrics Bull.*, **31**, 405–14.

Wold, H. (1954). Causality and econometrics. *Econometrica*, **22**, 162–77.

Wright, S. (1921). Correlation and causation. *Journal of Agricultural Research*, **20**, 557–85.

Yule, G. U. (1912). On the methods for measuring association between two attributes. *Journal of Royal Statistics Society*, **75**, 579–642.

Zanella, A. (2001). Measures and models of customer satisfaction: the underlying conceptual construct and comparison of different approaches. In *Proceedings of The 6th TQM World Congress.* Saint Petersburg.

Appendices

Appendix A

A suite of R functions

In this Appendix we provide a suite of R functions developed for learning network structure from data. In detail, functions concern (1) the setting of variable type; (2) tests for checking conditional independence; (3) PC algorithm; (4) OPC algorithm; (5) NOPC algorithm.

A.1 Setting the type of variable

When the set of data contains all nominal variables we can set each variables of the dataset as a factor. The function appropriate for this purpose is `factor.data`.

Function `factor.data`: this function permits to set each variable of dataset as factor.

Argument: **dataset**, a data frame containing the variables of interests.

Output: a data frame which columns are factors.

```
factor.data<-function(dataset){
        new<-c(1:nrow(dataset))
        for(i in 1:ncol(dataset)) {
                dataset[,i]<-factor(dataset[,i], ordered=F)
                new<-data.frame(new,dataset[,i])
                }
        new<-new[,-1]; colnames(new)<-names(dataset)
        return(new)
}
```

A.1 Setting the type of variable

When the set of data contains all ordinal variables we can set each variable of the dataset as an ordered factor. The function appropriate for this purpose is `ordered.data`.

Function `ordered.data`: this function permits to set each variable of dataset as ordered factor.

Argument: **dataset**, a data frame containing the variables of interests.

Output: a data frame which columns are ordered factors.

```
ordered.data<-function(dataset){
        new<-c(1:nrow(dataset))
        for(i in 1:ncol(dataset)) {
                dataset[,i]<-as.ordered(dataset[,i])
                new<-data.frame(new,dataset[,i])
                }
        new<-new[,-1]
        colnames(new)<-names(dataset)
        return(new)
}
```

When the set of data contains both nominal and ordinal variables we can set the right type of each variable. The function appropriate for this purpose is `SetVarType`.

Function `SetVarType`: this function permits to set the type of each variable by means of an editor window.

Argument: **dataset**, a data frame containing the variables of interests.

Output: a data frame which columns are either factors or ordered factors.

```
SetVarType<-function(dataset){
        list<-edit(data.frame(node=colnames(dataset), varType=c(rep
        ("Factor",(length(dataset)-1)),rep("Ordered", 1))))
        varType<-list[,2]
        new<-c(1:nrow(dataset))
                for(i in 1:nrow(list)) {
                if(varType[i]=="Ordered") dataset[,i]<-as.ordered(dataset[,i])
                else dataset[,i]<-factor(dataset[,i], ordered=F)
                new<-data.frame(new,dataset[,i])
                }
        new<-new[,-1]
        colnames(new)<-list[,1]
        return(new)
}
```

A.2 Conditional Independence tests

The PC algorithm checks conditional independence between variables computing the conditional cross entropy, known as mutual information. The `R` function for computing the conditional cross entropy has been declared `cond.cross.entropy`.

Function `cond.cross.entropy`: this function permits to compute the conditional cross entropy between a pair of variables given a conditioning set.

Arguments: **m**, an ftable; **d1** and **d2**, dimensions of the table.

Output: the value of conditional cross entropy CCE; the G^2 statistic; the degrees of freedom; the p-value.

```
cond.cross.entropy<-function(m,d1,d2){
        m[m==0]<-1e-10
                cross.entropy<-function(t,d1,d2){
                dim(t)<-c(d1,d2); t1<-addmargins(t)
                cm <- t1[d1+1,1:d2]; rm <- t1[1:d1,d2+1]; N<- t1[d1+1,d2+1]
                cm[cm==0]<-1e-10; rm[rm==0]<-1e-10; N[N==0]<-1e-10
                pr<-t/N; pcm<-c(cm/N); prm<-c(rm/N)
                pr[pr==0]<-1e-10
                dim(pcm)<-c(1,d2); dim(prm)<-c(d1,1)
                kl<-prm%*%pcm
                kl[kl==0]<-1e-10
                CE<- sum(pr*log(pr/kl))
                }
        ans<-apply(m,1,cross.entropy,d1,d2)
        fcond<-addmargins(m)[1:dim(m)[1],dim(m)[2]+1]
        tot<-addmargins(m)[dim(m)[1]+1,dim(m)[2]+1]
        CCE<-sum((fcond/sum(fcond))*(ans))
        dof<-(d1-1)*(d2-1)*(dim(m)[1]); g2<-2*tot*CCE ;pvalue<-pchisq(g2,lower.tail=F,df=dof)
        return(list(CCE=CCE, df=dof,g2=g2,P=pvalue))
}
```

A.3 PC algorithm

Here, we provide a function, namely `PC`, for computing the PC algorithm. The function is a variation of an already existing function called `pcAlgo` in the `pcalg` package of `R`. For this reason it requires the library `pcalg`.

Function `PC`: this function permits to compute the PC algorithm given a nominal dataset. The variables can be set as factors by means of the `factor.data` function.

Arguments: **dm**, a data frame; **n**, number of observations; **alpha**, significance level (default 0.05); **verbose**, a logical argument for obtaining a short output (default verbose=FALSE) or a detailed output (verbose=TRUE); **G**, the adjacency matrix; **NAdelete**, a logical argument for deleting edges when the p-value is not available (default NAdelate=TRUE).

Output: it is an object of a customized class called `myPCalgo`. The short output provides the fitted skeleton of the graph; the detailed output provides results of each test, the adjacency matrix of the final graph and the fitted skeleton.

```
PC<-function (dm = NA, n = NA, alpha=0.05, verbose = FALSE,
    G = NULL, NAdelete = TRUE) {
    if (any(is.na(dm))) {
        stopifnot(!is.na(n) > 0)
    }
    else {
        n <- nrow(dm)
        p <- ncol(dm)
    }
    n <- as.integer(n)
    cl <- match.call()
    sepset <- vector("list", p)
    n.edgetests <- numeric(1)
    if (is.null(G)) {
        G <- matrix(rep(TRUE, p * p), nrow = p, ncol = p)
        diag(G) <- FALSE
    }
    else {
       if (!(identical(dim(G), c(p, p)))) {
            stop("Dimensions of the dataset and G do not agree.")
        }
    }
    seq_p <- 1:p
    for (iList in 1:p) sepset[[iList]] <- vector("list", p)
    done <- FALSE
    ord <- 0
            dm.df <- as.data.frame(dm)
            while (!done && any(G) && ord <= Inf) {
                n.edgetests[ord + 1] <- 0
                done <- TRUE
                ind <- which(G, arr.ind = TRUE)
                ind <- ind[order(ind[, 1]), ]
                ind <- subset(ind, ind[,2]>ind[,1])
                remainingEdgeTests <- nrow(ind)
                if (verbose)
                  cat("Order=", ord, "; remaining edges:",
                  remainingEdgeTests, "\n", sep = "")
                for (i in 1:remainingEdgeTests) {
                  x <- ind[i, 1]
                  y <- ind[i, 2]
                  if (G[y, x]) {
                    nbrsBool <- G[, x]
                    nbrsBool[y] <- FALSE
                    nbrs <- seq_p[nbrsBool]
                    length_nbrs <- length(nbrs)
                    if (length_nbrs >= ord) {
                      if (length_nbrs > ord)
                        done <- FALSE
                      S <- seq(length = ord)
                      repeat {
                        n.edgetests[ord + 1] <- n.edgetests[ord +
                          1] + 1
prob <- CI.discrete(x, y, nbrs[S], dm.df, test="CCE")$P
T<-CI.discrete(x, y, nbrs[S], dm.df, test="CCE")$g2
if (verbose)
                          cat("x=", x, " y=", y, " S=", nbrs[S],
"p-value =", prob, ifelse(prob<alpha,"dip","---"),"\n", sep="\t")
                        if (is.na(prob))
                          prob <- ifelse(NAdelete, 1, 0)
                        if (prob >= alpha) {
                          G[x, y] <- G[y, x] <- FALSE
                          sepset[[x]][[y]] <- nbrs[S]
                          break
                        }
                        else {
                          nextSet <- getNextSet(length_nbrs,
                            ord, S)
                          if (nextSet$wasLast)
                            break
                          S <- nextSet$nextSet
                        }
                      }
                    }
                  }
                }
                ord <- ord + 1
            }
    if (verbose) {
        cat("Final graph adjacency matrix:\n")
        print(adjMatrix(G))
    }
    if (sum(G) == 0) {
        Gobject <- new("graphNEL", nodes = as.character(seq_p))
    }
    else {
        colnames(G) <- rownames(G) <- as.character(seq_p)
        Gobject <- as(G, "graphNEL")
    }
   setClass("oldPCalgo", contains="pcAlgo", "VIRTUAL")
   setClass("myPCalgo", representation(graph="graph", adj="matrix"),
   contains="oldPCalgo")
   res <- new("myPCalgo", graph = Gobject, adj=adjMatrix(G), call = cl, n = n,
   max.ord = as.integer(ord-1), n.edgetests = n.edgetests, sepset = sepset)
   res
}
```

A.4 OPC algorithm

Here, we provide a function, namely `OPC`, for computing the OPC algorithm. The structure is similar to `PC` function; it requires the library `pcalg`.

Function `OPC`: this function permits to compute the OPC algorithm from an ordinal dataset. The variables must be ordinal. We can set of data frame as ordered factors by means of the `ordered.data` function.

Arguments: **dm**, a data frame; **n**, number of observations; **alpha**, significance level (default 0.05); **verbose**, a logical argument for obtaining a short output (default verbose=FALSE) or a detailed output (verbose=TRUE); **G**, the adjacency matrix; **NAdelete**, a logical argument for deleting edges when the p-value is not available (default NAdelate=TRUE).

Output: it is an object of a customized class called `myPCalgo`. The short output provides the fitted skeleton of the graph; the detailed output provides results of each test, the adjacency matrix of the final graph and the fitted skeleton.

```
OPC<-function (dm = NA, n = NA, alpha=0.05, verbose = FALSE,
    G = NULL, NAdelete = TRUE)
{
    if (any(is.na(dm))) {
        stopifnot(!is.na(n) > 0)
    }
    else {
        n <- nrow(dm)
        p <- ncol(dm)
    }
    n <- as.integer(n)
    cl <- match.call()
    sepset <- vector("list", p)
    n.edgetests <- numeric(1)
```

```
if (is.null(G)) {
    G <- matrix(rep(TRUE, p * p), nrow = p, ncol = p)
    diag(G) <- FALSE
}
else {
   if (!(identical(dim(G), c(p, p)))) {
        stop("Dimensions of the dataset and G do not agree.")
    }
}
seq_p <- 1:p
for (iList in 1:p) sepset[[iList]] <- vector("list", p)
done <- FALSE
ord <- 0
        dm.df <- as.data.frame(dm)
        while (!done && any(G) && ord <= Inf) {
            n.edgetests[ord + 1] <- 0
            done <- TRUE
            ind <- which(G, arr.ind = TRUE)
            ind <- ind[order(ind[, 1]), ]
            ind <- subset(ind, ind[,2]>ind[,1])
            remainingEdgeTests <- nrow(ind)
            if (verbose)
              cat("Order=", ord, "; remaining edges:",
              remainingEdgeTests, "\n", sep = "")
            for (i in 1:remainingEdgeTests) {
              x <- ind[i, 1]
              y <- ind[i, 2]
              if (G[y, x]) {
                nbrsBool <- G[, x]
                nbrsBool[y] <- FALSE
                nbrs <- seq_p[nbrsBool]
                length_nbrs <- length(nbrs)
                if (length_nbrs >= ord) {
                  if (length_nbrs > ord)
                    done <- FALSE
                  S <- seq(length = ord)
                  repeat {
                    n.edgetests[ord + 1] <- n.edgetests[ord +
                      1] + 1
if (!(class(dm.df[,x])[1]=="ordered" &
class(dm.df[,y])[1]=="ordered")) {
stop("invalide procedure")
}
prob <- CI.discrete(x, y, nbrs[S], dm.df, test="jt")$P
T<-CI.discrete(x, y, nbrs[S], dm.df, test="jt")$JT
if (verbose)
              cat("x=", x, " y=", y, " S=", nbrs[S],
"p-value =", prob, ifelse(prob<alpha,"dip","---"),"\n", sep="\t")
                        if (is.na(prob))
                          prob <- ifelse(NAdelete, 1, 0)
                        if (prob >= alpha) {
                          G[x, y] <- G[y, x] <- FALSE
                          sepset[[x]][[y]] <- nbrs[S]
                          break
                        }
                        else {
                          nextSet <- getNextSet(length_nbrs,
                            ord, S)
                          if (nextSet$wasLast)
                            break
                          S <- nextSet$nextSet
                        }
                      }
                    }
                  }
                }
                ord <- ord + 1
            }
    if (verbose) {
        cat("Final graph adjacency matrix:\n")
        print(adjMatrix(G))
    }
    if (sum(G) == 0) {
        Gobject <- new("graphNEL", nodes = as.character(seq_p))
    }
    else {
        colnames(G) <- rownames(G) <- as.character(seq_p)
        Gobject <- as(G, "graphNEL")
    }
   setClass("oldPCalgo", contains="pcAlgo", "VIRTUAL")
   setClass("myPCalgo", representation(graph="graph", adj="matrix"),
    contains="oldPCalgo")
   res <- new("myPCalgo", graph = Gobject, adj=adjMatrix(G),
    call = cl, n = n, max.ord = as.integer(ord - 1),
    n.edgetests = n.edgetests, sepset = sepset)
   res
}
```

A.5 NOPC algorithm

Here, we provide a function, namely `NOPC`, for computing the NOPC algorithm. The procedure selects the most appropriate test for checking conditional independence according to the type of variables. In order to set the type of each variable, the `SetVarType` function can be used. The `NOPC` function requires the library `pcalg`.

Function `NOPC`: this function permits to compute the NOPC algorithm from a dataset made of mixed nominal-ordinal variables.

Arguments: **dm**, a data frame; **n**, number of observations; **alpha**, significance level (default 0.05); **verbose**, a logical argument for obtaining a short output (default verbose=FALSE) or a detailed output (verbose=TRUE); **G**, the adjacency matrix; **NAdelete**, a logical argument for deleting edges when the p-value is not available (default NAdelate=TRUE).

Output: it is an object of a customized class called `myPCalgo`. The short output provides the fitted skeleton of the graph; the detailed output provides results of each test, the adjacency matrix of the final graph and the fitted skeleton.

```
NOPC<-function (dm = NA, n = NA, alpha=0.05, verbose = FALSE,
    G = NULL, NAdelete = TRUE)
{
    if (any(is.na(dm))) {
        stopifnot(!is.na(n) > 0)
    }
    else {
        n <- nrow(dm)
        p <- ncol(dm)
    }
    n <- as.integer(n)
    cl <- match.call()
    sepset <- vector("list", p)
    n.edgetests <- numeric(1)
    if (is.null(G)) {
        G <- matrix(rep(TRUE, p * p), nrow = p, ncol = p)
        diag(G) <- FALSE
    }
    else {
       if (!(identical(dim(G), c(p, p)))) {
            stop("Dimensions of the dataset and G do not agree.")
        }
    }
    seq_p <- 1:p
    for (iList in 1:p) sepset[[iList]] <- vector("list", p)
    done <- FALSE
    ord <- 0
            dm.df <- as.data.frame(dm)
            while (!done && any(G) && ord <= Inf) {
                n.edgetests[ord + 1] <- 0
                done <- TRUE
                ind <- which(G, arr.ind = TRUE)
                ind <- ind[order(ind[, 1]), ]
                ind <- subset(ind, ind[,2]>ind[,1])
                remainingEdgeTests <- nrow(ind)
                if (verbose)
                  cat("Order=", ord, "; remaining edges:", remainingEdgeTests,
                    "\n", sep = "")
                for (i in 1:remainingEdgeTests) {
                  x <- ind[i, 1]
                  y <- ind[i, 2]
                  if (G[y, x]) {
                    nbrsBool <- G[, x]
                    nbrsBool[y] <- FALSE
                    nbrs <- seq_p[nbrsBool]
                    length_nbrs <- length(nbrs)
                    if (length_nbrs >= ord) {
                      if (length_nbrs > ord)
                        done <- FALSE
                      S <- seq(length = ord)
                      repeat {
                        n.edgetests[ord + 1] <- n.edgetests[ord +
                          1] + 1
if (class(dm.df[,x])[1]=="ordered" & class(dm.df[,y])[1]=="ordered"){
prob <- CI.discrete(x, y, nbrs[S], dm.df, test="jt")$P
T<-CI.discrete(x, y, nbrs[S], dm.df, test="jt")$JT
} else{
if (class(dm.df[,x])[1]=="factor" & class(dm.df[,y])[1]=="ordered"
& nlevels(dm.df[,x])==2) {
prob <- CI.discrete(x, y, nbrs[S], dm.df, test="wilcoxon")$P
T<-CI.discrete(x, y, nbrs[S], dm.df, test="wilcoxon")$W
} else{
if (class(dm.df[,x])[1]=="factor" & class(dm.df[,y])[1]=="ordered") {
prob <- CI.discrete(x, y, nbrs[S], dm.df, test="kruskal")$P
T<-CI.discrete(x, y, nbrs[S], dm.df, test="kruskal")$KW
} else{
if (class(dm.df[,x])[1]=="factor" & class(dm.df[,y])[1]=="factor") {
prob <- CI.discrete(x, y, nbrs[S], dm.df, test="CCE")$P
T<-CI.discrete(x, y, nbrs[S], dm.df, test="CCE")$g2
} else{
prob <- CI.discrete(x, y, nbrs[S], dm.df, test="CCE")$P
T<-CI.discrete(x, y, nbrs[S], dm.df, test="CCE")$g2}
}}}
if(prob=="NaN"){
prob <- CI.discrete(x, y, nbrs[S], dm.df, test="CCE")$P
T<-CI.discrete(x, y, nbrs[S], dm.df, test="CCE")$g2
}
if (verbose)
                          cat("x=", x, " y=", y, " S=", nbrs[S],
"p-value =", prob, ifelse(prob<alpha,"dip","---"),"\n", sep="\t")
                        if (is.na(prob))
                          prob <- ifelse(NAdelete, 1, 0)
                        if (prob >= alpha) {
                          G[x, y] <- G[y, x] <- FALSE
                          sepset[[x]][[y]] <- nbrs[S]
```

```
             break
            }
            else {
             nextSet <- getNextSet(length_nbrs,
              ord, S)
             if (nextSet$wasLast)
              break
             S <- nextSet$nextSet
            }
           }
          }
         }
        }
        ord <- ord + 1
      }
  if (verbose) {
    cat("Final graph adjacency matrix:\n")
    print(adjMatrix(G))
  }
  if (sum(G) == 0) {
    Gobject <- new("graphNEL", nodes = as.character(seq_p))
  }
  else {
    colnames(G) <- rownames(G) <- as.character(seq_p)
    Gobject <- as(G, "graphNEL")
  }
 setClass("oldPCalgo", contains="pcAlgo", "VIRTUAL")
 setClass("myPCalgo", representation(graph="graph", adj="matrix"), contains="oldPCalgo")
 res <- new("myPCalgo", graph = Gobject, adj=adjMatrix(G), call = cl, n = n,
  max.ord = as.integer(ord -
    1), n.edgetests = n.edgetests, sepset = sepset)
 res
}
```

Appendix B

R code

In this Appendix we provide the `R` code used for comparing performance of PC and OPC algorithm.

Load packages.

```
library(RHugin)
library(pcalg)
```

Choose functions path.

```
path=choose.dir(getwd(), "Directory containing functions")
flist<-list.files(path)
for (i in 1:length(flist)) source(paste(path,"\\",flist[i],sep=""))
```

Load and compile the real network. Here, the network is a Hugin object.

```
network<-read.rhd("file.net",type="net")
compile.RHuginDomain(network)
```

Use a more suitable class graph and build the real skeleton of the DAG.

```
tDAG<-as.graph.RHuginDomain(network)
RealSkel<-ugraph(tDAG)
```

Build an adjacency matrix representing the real PDAG. Firstly, build a matrix from the real skeleton, then change manually elements of matrix according to DAG v-structures. Define the generic element a_{ij} in the matrix as 0 if the arrow from node j to node i is compelled. Finally, provide a graphical representation of the PDAG using an object of class graph.

```
matrixPDAG<-as(RealSkel, "matrix")
matrixPDAG[i,j]<-0
.....
RealPDAG<-as(matrixPDAG, "graphNEL")
```

Plot the network in R.

```
library(igraph)
Vertex.names<-c(get.nodes(network))
Real_DAG<-igraph.from.graphNEL(tDAG)
tkplot(Real_DAG, vertex.label=Vertex.names, vertex.size=50, vertex.size2=20)
detach("package:igraph")
```

Simulate k samples with n cases from the real network. Here, $k = 1000$ and $n = 50$. Set a seed s.

```
k<-1000
n<-n
set.seed(s)
seeds<-sample(1:(k*100),k)
data<-list()
for (i in 1:k){
data[[i]]<-as.data.frame(simulate.RHuginDomain(network,n, seeds[[i]]))
}
```

Apply to each generated dataset the PC function to estimate the skeleton by PC algorithm.

```
PCskel<-list()
for(i in 1:k){
data[[i]]<-factor.data(data[[i]])
PCskel[[i]]<-PC(data[[i]])
}
```

Find the PDAG of each learnt structure by means of function udag2pdag of pcalg package.

```
PCpdag<-list()
for(i in 1:k){
PCpdag[[i]]<-udag2pdag(PCskel[[i]])
}
```

Apply to each generated dataset the OPC function to estimate the skeleton by OPC algorithm.

```
OPCskel<-list()
for(i in 1:k){
data[[i]]<-ordered.data(data[[i]])
OPCskel[[i]]<-OPC(data[[i]])
}
```

Find the PDAG of each learnt structure by means of function `udag2pdag` of `pcalg` package.

```
OPCpdag<-list()
for(i in 1:k){
OPCpdag[[i]]<-udag2pdag(OPCskel[[i]])
}
```

Evaluate the accuracy of algorithms. Firstly, compare the skeleton of each learnt structure with the real skeleton and compute TPR, FPR and TDR. Store the results as vectors and compute the mean values for each index.

```
comparison_PC_50<-list()
comparison_OPC_50<-list()
for (i in 1:k){
comparison_PC_50[[i]]<-compareGraphs(PCskel[[i]]@graph, RealSkel)
comparison_OPC_50[[i]]<-compareGraphs(OPCskel[[i]]@graph, RealSkel)
}

comparison_PC.50<-matrix(nrow=k,ncol=3)
for (i in 1:k){
comparison_PC.50[i,]<-as.vector(comparison_PC_50[[i]])
}
dimnames(comparison_PC.50)<-list(c(),c("TPR", "FPR", "TDR"))
comparison_PC_50<-apply(comparison_PC.50,2,mean)
comparison_PC_50

comparison_OPC.50<-matrix(nrow=k,ncol=3)
for (i in 1:k){
comparison_OPC.50[i,]<-as.vector(comparison_OPC_50[[i]])
}
dimnames(comparison_OPC.50)<-list(c(),c("TPR", "FPR", "TDR"))
comparison_OPC_50<-apply(comparison_OPC.50,2,mean)
comparison_OPC_50
```

Compute the Structural Hamming Distance for each fitted PDAG.

```
shd_PC.50<-list()
shd_OPC.50<-list()
for(i in 1:k){
shd_PC.50[i]<-shd(RealPDAG,PCpdag[[i]]@graph)
shd_OPC.50[i]<-shd(RealPDAG,OPCpdag[[i]]@graph)
}
shd_PC_50<-matrix(nrow=1,ncol=k)
for (i in 1:k){
shd_PC_50[,i]<-as.vector(shd_PC.50[[i]])
}
dimnames(shd_PC_50)<-list(c("SHD"), c())
shd_PC_50<-apply(shd_PC_50,1,mean)
shd_PC_50

shd_OPC_50<-matrix(nrow=1,ncol=k)
for (i in 1:k){
shd_OPC_50[,i]<-as.vector(shd_OPC.50[[i]])
}
dimnames(shd_OPC_50)<-list(c("SHD"), c())
shd_OPC_50<-apply(shd_OPC_50,1,mean)
shd_OPC_50
```

Repeat the code using $n = 100$ and $n = 500$. Store results of skeletons comparison in vectors properly labelled (e.g. `comparison_PC_100`, `comparison_PC_500`, `comparison_PC_100`, `comparison_PC_500`). Save results of SHD using the appropriate labels (e.g. `shd_PC_100`, `shd_PC_500`, `shd_PC_100`, `shd_PC_500`).

Compare the results and plot the SHD performance.

```
plot(shd_PC, type="b", col="red",main="Structural distance",
xlab="sample size", ylab="SHD ave", lwd=2, ylim=c(0,7), xaxt="n")
axis(1, 1:3, c(50,100,500)); lines(shd_OPC,col="blue", type="b", lwd=2)
legend("topright", legend=c("PC-algorithm", "OPC-algorithm"), col=c("red","blue"), lwd=3)

par(mfrow=c(1,2))
boxplot(as.numeric(shd_PC.50), as.numeric(shd_PC.100), as.numeric(shd_PC.500),
names=c(50,100,500),xlab="Sample size", outline=T)
title("Boxplot of SHD - PC algorithm")
boxplot(as.numeric(shd_OPC.50), as.numeric(shd_OPC.100), as.numeric(shd_OPC.500),
names=c(50,100,500),xlab="Sample size", outline=T)
title("Boxplot of SHD - OPC algorithm")
```

List of Figures

List of Tables