View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Padua@thesis

Università degli Studi di Padova
Dipartimento di Ingegneria dell'Informazione

Master Degree in Computer Engineering
a.y. 2013/14

An Integer Programming approach to Bayesian Network Structure Learning

Alberto Franzin

Advisor:
Prof. Silvana Badaloni

Co-advisor:
Dr. Francesco Sambo

April 14th, 2014

Abstract

In this thesis we study the problem of learning a Bayesian Network structure from data using an Integer Programming approach. Bayesian Networks are Probabilistic Graphical Models that represent causality relationships among variables, in the form of a Directed Acyclic Graph (DAG). One of the open problems regarding Bayesian Networks is how to infer the DAG of the network from a dataset. We study the existing approaches, an in particular some recent works that formulate the problem as an Integer Programming (IP) model. By discussing some weaknesses of the existing approaches, we propose an alternative model and a different solution, based on a statistical sparsification of the search space. We discuss our model from a theoretical point of view, and evaluate it against other IP-based software packages. Results show how our approach can lead to promising results, especially for large networks.

Contents

List of Figures v

List of Tables vii

1 **Introduction** 1

2 **Theoretical foundations** 5
- 2.1 General graph theory 5
- 2.2 Probability theory 7
  - 2.2.1 Hypothesis testing 9
  - 2.2.2 Information theory 10
- 2.3 Bayesian Networks 12
  - 2.3.1 Definition 12
  - 2.3.2 BNs and conditional independence 14
  - 2.3.3 Properties 17
- 2.4 Mixed-Integer Linear Programming 18
  - 2.4.1 Theoretical introduction 19
  - 2.4.2 Techniques for solving MILPs 20
  - 2.4.3 Software for solving MILPs 24

3 **Bayesian Network Structure Learning** 29
- 3.1 Scoring metrics 30
  - 3.1.1 General properties 31
  - 3.1.2 Bayesian scoring functions 32
  - 3.1.3 Information theoretic scoring functions 34
- 3.2 Independence tests 37
- 3.3 Algorithms 38
  - 3.3.1 Dynamic Programming 38
  - 3.3.2 Greedy 39

3.3.3 Max-min hill-climbing 40
3.3.4 Branch-and-bound 41
3.3.5 Local learning 43
3.3.6 Structure learning as IP problem 44
3.3.7 Hybrid methods and other approaches 49
3.4 Comments 51
3.4.1 On the significance of the results 54

4 Alternative Integer Programming formulation 57
4.1 Reducing the search space 58
4.2 From sets of nodes to edges 59
4.3 A family of skeleton-based cuts 64
4.3.1 Finding violated cuts 65
4.4 Notes on the model 65
4.4.1 Relations with other problems 67
4.5 A cutting-plane algorithm 70
4.5.1 Computational costs and considerations 71
4.5.2 Finding cycles 73
4.6 Solving the model efficiently 77
4.6.1 Partial linear relaxation 77
4.6.2 Computational techniques 80

5 Experimental results 85
5.1 Preliminary considerations 86
5.2 Test description 87
5.3 Results 89
5.3.1 Successful tests 89
5.3.2 Score of the networks 92
5.3.3 Structural Hamming Distance 95
5.3.4 Time performances 95
5.3.5 Memory allocation 102
5.4 Comments 105

6 Conclusions 111
6.1 Future directions 113

Bibliography 117

List of Figures

2.1 Flows on influence in a BN 14
2.2 Example of equivalence classes for a simple graph 18
2.3 A polytope in $\mathbb{R}^2$ 21
2.4 Comparison of MIP solvers with respect to SCIP 26

3.1 Parent set lattice for 4 variables 30
3.2 Orientation rules for patterns 50

4.1 Cycles over same set of edges 71
4.2 Steps of (k, l)-BFS 76
4.3 Degenerate case for (k, l)-BFS 76
4.4 Geometric effect of callbacks 83

5.1 Failure for HEPAR2_100 91
5.2 Score results 94
5.3 SHD results 96
5.4 Preprocessing time results 99
5.5 Edge orientation time results 100
5.6 Overall time results 101
5.7 Memory results 103
5.8 Evolution of ALARM_10000 105

List of Tables

2.1 Summary of flow of influence in BNs 17

3.1 Summary of structure learning algorithms and strategies . 52

5.1 In-degree of the instances 88
5.2 Summary of successful results 90
5.3 Summary of score results 93
5.4 Summary of SHD results 97
5.5 Summary of time results 98
5.6 Summary of memory results 104

Introduction 1

Probabilistic Graphical Models (PGMs) are a powerful way of representing relationships of conditional dependence among a set of variables. Variables are associated to nodes in a graph, and edges between nodes denote the relations of conditional dependence. Because of the succint yet intuitive representation, the power of the framework, and the variety of applications, PGMs have been used to address a large set of problems, from coding theory to biology, from computer vision to diagnostic, via classification, language processing, and much more.

Graphical Models can be divided in two categories: Markov Networks, if the graph is undirected and possibly cyclic, and Bayesian Networks, when the graph is directed and acyclic. Markov and Bayesian Networks share many common properties, with some differences, in particular those concerning the representable dependencies that the other model cannot encode.

A thorough introduction on Probabilistic Graphical Models can be found in Koller and Friedman [67].

Several problems concerning PGMs are, however, still open. One among those open questions is how to build a Bayesian Network from a set of data: in its general form, the problem has been proven to be NP-complete [17]. This problem has been studied extensively, and is still an active field of research. Because of the hardness of this task, there is still no easy way of learning a graphical model from a dataset. Many exact and approximate methods have been proposed, but up to

now there is no algorithms which can build an optimal network for big enough problems, or even a sufficiently decent one for large size instances.

In this thesis, we study the problem of learning a Bayesian Network from data using mixed-integer programming techniques. Specifically, we address the issue of discovering a plausible Bayesian Network structure by modeling the problem as an integer program. We study some recent integer programming approaches that have been proposed in the literature ([57, 27, 54] and related works), and move from some observations to them in order to provide a more robust approach. The common approach to the problem is based on *scoring* the candidate *components* of the network according to the dataset. Those components are the possible configurations of parent nodes for the nodes in the graph; the score is instead computed according to the given data. The selected components have to contribute to the maximum score possible for the overall collection while forming an acyclic graph. The two main issues of this formulation are the exponential growth of the number of candidates with respect to the number of variables observed, and the difficulty entailed by the request of acyclicity of the resulting network.

Among the proposals for mitigating these hurdles, one commonly adopted technique is to assume a bounded number of parents for each node. Clearly this bound, while maintaining under control the computational resources needed to perform the full evaluation, in general may discard configurations that belong to the optimal solution. More importantly, this sparsification is done blindly, in a way that completely ignores the real structure of the observed data. We discuss more deeply the implications of this pathway, and choose instead to follow a different approach, namely applying an early statistical test in order to discover an underlying undirected graph, following a path developed in [120]. We therefore aim to reduce the number of candidates to evaluate in a more sensible manner, selecting the candidate sets to be discarded by looking at the data. The task therefore becomes to find an edge orientation of the newfound undirected graph that is consistent with the initial requirements of maximizing the overall score of the final network and avoiding directed cycles.

We propose an integer programming formulation for the problem formulated this way, relating the candidate parent sets to the edge orientation they entail. We subsequently evaluate this approach against

some preexistent methods. It has to be said that, while our model is complete, the way we choose to solve it also yields a loss of information that may prevent the optimal solution to be found; however, we show that it is a more reasonable strategy than its alternative when the size of the network grows up to some dozens of nodes, in particular when there is enough data to perform valid statistical inferences.

This thesis is divided as follows: in the second chapter, Bayesian Networks are defined as mathematical objects, along with their theoretical foundations, then a brief introduction to integer programming is given. In the third chapter, we review the state-of-the-art approaches for learning Bayesian Networks. In the fourth chapter we present our contribution to the problem. In the fifth chapter we show the experimental results we have obtained, and finally we expose our conclusions and some possible directions for new research along this path.

Theoretical foundations 2

In this chapter we introduce the theoretical basis of Bayesian Networks. In order to provide a sufficient framework for the discussion, we first briefly present the two areas Bayesian Networks lies within, namely graph theory and probability theory. Then, we provide some background on mixed-integer linear programming, as its tools are used in the main contributions of this thesis.

2.1 General graph theory

Graph theory is a branch of mathematics well studied and widely used in many areas of theoretical and applied sciences. Its concepts, definitions, and properties can be found in plenty of books and textbooks, see for example Bondy and Murty [13], Bollobás [12], West [122]. Here we report some concepts used thoughout the rest of this work, in order to provide a shared vocabulary for the reader.

A *graph* $G = (V, E)$ is a pair of sets, namely the set of *nodes* V, and the set of *edges* $E \subseteq V \times V$ connecting two nodes. When two nodes can be directly connected via more than one edge, such a graph is called a *multigraph*; when multigraphs are possible, graphs with no multiple edges are called *simple graph* in order to distinguish them. In the remainder of this work, multigraphs are not considered, so we will use the term *graph* to indicate only simple ones, since there

is no risk of confusion. A graph is an *undirected graph* if $\forall e \in E$, $e = (u, v) = (v, u)$, that is, every edge can be traversed both ways. When, instead, every edge (u, v) in E allows to go from node u to node v but not viceversa, we call it a *directed graph*; u is the *tail* of the edge, while v is its *head*. To avoid confusion, we will use the notation (u, v) to indicate undirected edges, and $(u \rightarrow v)$ to denote directed edges, or *arcs*. A graph including both directed and undirected edges is called a *partially directed graph*. A graph G obtained by removing some nodes or edges from another graph H is called a *subgraph* of H.

Two nodes connected by an edge are said to be *adjacent*. A sequence of nodes $[u_1, u_2, \ldots, u_n]$ is called an *undirected path* from u_1 to u_n. If all the nodes are different, the path is called *simple path*. If $u_1 = u_n$, such a path is called an *undirected cycle*; if $u_1 = u_n$ is the only repeatedly visited node the cycle is a *simple cycle*. If the sequence of edges connecting the nodes in the given order is composed of directed edges, the previous concepts exist as well, assuming the names of, respectively, *directed path* and *directed cycle*. From here onwards, the terms *path* and *cycle* will denote only simple paths and simple cycles, both directed and undirected. A graph without cycles is called an *acyclic graph*. An acyclic graph G is also called a *forest*; if G is connected, it is a *tree*. A node u_1, or a set of nodes U_1, is *connected* to a node $u_2 \neq u_1$, or to a set of nodes U_2 such that $U_1 \cap U_2 = \emptyset$, if there is a path linking them. In directed graphs, or in undirected graphs whenever a directionality notion arises (e.g. a traversal), a node u can be defined as a *parent* for another node v if an edge going from u to v exists in the graph; conversely, in the same context v is a *child node* of u. If every couple of nodes in a graph G are connected via some path, also G is a *connected graph*.

A graph where $E = V \times V$ is said to be *complete*. The opposite, that is a graph where $E = \emptyset$ is an *empty graph*. A graph with a number of edges close to the maximum possible number is called a *dense graph*, while a graph with only "few" edges is called a *sparse graph*. Such notions are vague, as the category which a graph falls in usually depends on the context; many times, a graph is sparse if $|E| \in O(|V|) or O(\log |V|)$, while a dense graph has $O(|V|^2)$ edges. The number of edges insisting on a node v is the *degree* of v, and we denote it with $\deg(v)$. The highest degree among all of the nodes in a graph G also determines the *degree of graph* G. We will use $\delta(v)$ to indicate the set of edges insisting on a node v. For the set of incoming

and outgoing edges of v, we will write $\delta^-(v)$ and $\delta^+(v)$, respectively; in case of undirected graphs, those two sets coincide. $\delta(v)$ is called a *cut* for node v, because removing its nodes render v disconnected from the rest of the graph. The same notion extends to sets of nodes.

A graph is a *planar graph* if it can be drawn on an euclidean plane without overlapping edges. An *eulerian path* is a path that traverses once and only once every edge in the graph. An eulerian path that returns back to the starting node is an *eulerian cycle*. A *hamiltonian path* is, instead, a path that traverses all of the nodes in the graph. If it returns to the starting node, it is called a *hamiltonian cycle*. A *clique* is a maximal connected subcycle, that is, it forms a complete subgraph, and there are no other nodes in the graph that can be added to the clique while maintaining the completeness property.

A *flag* is an induced subgraph representing a path that contains both directed and undirected edges: for example, if $i, j, k \in V$, the sequence $((i \rightarrow j), (j, k))$ is a flag. A *chain graph* G is a graph that admints both directed and undirected edges whose nodes can be sorted in a *chain*: that is, there is a sequence of sets of nodes $C_1, \ldots, C_m, m \geq 1$ such that, for every directed edge $(a \rightarrow b)$, $a \in C_i$, $b \in C_j$, $i < j$. Alternatively, a chain graph is a graph with no directed cycles. One immediate consequence is that both DAGs and undirected graphs are chain graphs; we define them as *chain graphs without flags*.

2.2 Probability theory

Probability theory is the branch of mathematics studying events under conditions of uncertainty or randomness. Again, we introduce some basic concepts employed in the remainder of this work. Specifically, here we deal only with discrete distributions. For more comprehensive references on theory and applications, one may refer to Ross [90], Bishop and Nasrabadi [10], among many books and textbooks.

In its discrete declination, probability theory is the subject of counting possibilities for an event to occur. The *probability of an event* may be defined as the ratio between favourable cases for the event to happen, and the number of possible cases. Let Ω be a countable set, called the *sample space*: we define the probability of an event $\omega \in \Omega$ to happen as a function $f(\omega) : \Omega \mapsto \mathbb{R}_{[0,1]}$, such that

$\sum_{\omega \in \Omega} f(\omega) = 1$. An *event* E is a subset of Ω; the probability of E is defined as $P(E) = \sum_{\omega \in E} f(\omega)$. The following properties hold: $P(\emptyset) = 0$, $P(\Omega) = 1$, and if $E \subseteq F$ then $P(E) \leq P(F)$.

We can define the following operations over the sample space. The *union of events* is defined as the probability that at least one of them occurs; it corresponds to the *or* logical operator. The *intersection of events*, corresponding to the logical operator *and* is instead defined as the probability that all of the events considered happen; the probability of two events E, F to happen simultaneously is called *joint probability*, and is written $P(E \cap F)$. The *complementary event* of an event E if the event $E^C = \Omega \setminus E$; from the previous definitions, it follows that $P(E^C) = 1 - P(E)$. The *expected value* $E[\cdot]$ or *mean* μ for a set of events X is defined as

$$E[X] = \sum_{x \in X} x \cdot p(x), \tag{2.1}$$

and the *variance* σ^2 is computed as $E[(X - \mu)^2]$.

Two events E, F are *independent events* if $P(E, F) = P(E)P(F)$, that is, if their intersection is the empty set. Conversely, the two events E, F are not independent if the occurrence of E influences the probability of F, and viceversa. We write $F \mid E$ to indicate that event F depends on event E. The *posterior probability* $P(F \mid E)$ is therefore the probability that F happens, knowing that E has happened. If $P(F \mid E) = P(F)$, and $P(E \mid F) = P(E)$, E and F are *conditionally independent*.

The *Bayes theorem* states that

$$P(E \mid F) = P(F \mid E) \frac{P(E)}{P(F)}, \tag{2.2}$$

meaning that the posterior probability of E conditioned on F can be computed by knowing the probability of F given E and the two *prior probabiities* of events E and F.

Any event that depends on chance can be modeled as a *random variable*, a variable that can take one value from a set of possible ones with some probability. Of course, the sum of all the probabilities of the values of A must sum to 1. The probability of each value a that A can assume is measured by the *probability density function* $p_A(a)$; the probabilities of a set of values $a_1, \ldots, a_k$ for A is instead measured

by the *probability distribution function* $F_A(a_k) = P(a_j : j \leq k)$. The complete set of possible assignments over the spece of events is called the *probability distribution*. The operations defined over events remain valid also for random variables.

Some probability distributions occur in many applications, and are therefore well known: among them, we mention the *Normal distribution* or *Gaussian distribution*, after C.F. Gauss:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.3}$$

for a random variable with mean μ and variance σ^2; a random variable X obeying this distribution is usually written as $X \sim \mathcal{N}$. Another distribution we'll mention in this thesis is the *Dirichlet distribution*, from J. Dirichlet, defined for a sequence $X_1, \ldots, X_k > 0$, $k \geq 2$, such that $X_1 + \cdots + X_k < 1$ with parameters $\alpha_1, \ldots, \alpha_k$:

$$p(x_1, \ldots, x_k) = c \cdot \prod_{i=1}^{k} x_i^{\alpha_i - 1}, \tag{2.4}$$

where c is a normalization factor. The last distribution we describe here is the χ^2 *distribution* with degree of freedom k, which models the distribution of the sum of a series of independent Normal random variables $X_1, \ldots, X_k$ as

$$Q = \sum_{i=1}^{k} x_i^2. \tag{2.5}$$

2.2.1 Hypothesis testing

When studying events, one of the key issues is to determine whether an event E has occurred just by chance, or, instead, there is some other event F that have caused it. This is the problem of inferring the *statistical significance* of E. We say that en event E is significant if is not likely that it has happened by chance alone; in other words, E is significant if the probability that it has occurred by chance alone is "low". This "low" is quantified in a threshold called the α *value*, which is the value of the probability for which we choose to consider E to be determined by chance alone or not.

In statistics, the assertion "event E is unrelated with respect to event F" is called the *null hypothesis* for E and F, and is generally denoted as H_0. Establishing whether H_0 is true or false is the subject of *statistical hypothesis testing*. This null hypothesis can be accepted (that is, E really depends only by chance) or rejected (E is instead related to F). The rejection if H_0, of course, requires some evidence to be shown. This evaluation is accomplished by performing statistical tests over the data, such as the χ^2 test, whose outcome is called the *p-value*, and corresponds to the probability of wrongly reject the null hypothesis, that is, to declare two events independent when, in reality, they are related. The p-value is then compared against the α threshold: if the p-value is less that α then the measured events are likely to be related, and the null hypothesis is rejected. Conversely, if the p-value is above the threshold α, then the null hypothesis is accepted, since there is no evidence that the relation among the two events is any stronger than pure chance. Wrongly rejecting H_0 is called *type I error*, or *false positive*, while by accepting a false null hypothesis one incurs in the *type II error*, or *false negative*.

Statistical hypothesis testing can be performed using methods such as the χ^2 *test*

$$\chi^2 = \sum_{x,y,\mathbf{z}} \frac{(N_{xy\mathbf{z}} - E_{xy\mathbf{z}})^2}{E_{xy\mathbf{z}}} \tag{2.6}$$

or the G^2 *test*

$$G^2 = 2 \sum_{x,y,\mathbf{z}} N_{x,y,\mathbf{z}} \ln \frac{N_{x,y,\mathbf{z}}}{E_{x,y,\mathbf{z}}} \tag{2.7}$$

for random variables X, Y, $\mathbf{Z}$ taking values respectively x, y, $\mathbf{z}$, and $E_{xy\mathbf{z}} = N_{x\mathbf{z}} N_{x\mathbf{z}} / N_{\mathbf{z}}$, whose asymptotic distribution is a χ^2 random variable. If F is the cumulative distribution function of the χ^2 random variable and $F(D)$ is its evaluation over a dataset D, the p-value is computed as $1 - F(D)$, and compared against the threshold α.

2.2.2 Information theory

Information theory is the science that studies the quantification of information. It was born in the mid of the past century with the seminal work of Shannon [97], who exploited the limits of the communication of a signal over a noisy channel. Information theoretical concepts oc-

cur in countless areas and applications of modern science. The reader may refer to Cover and Thomas [25] for a deeper introduction.

The basic idea of information theory is that information that occurs more frequently should be represented in a more succint way than infrequent information. The key concept is the *entropy* H of a set of signals $\mathbf{X}$, defined as

$$H(\mathbf{X}) = E_{\mathbf{X}}(I(x)) = -\sum_{x \in \mathbf{X}} p(x) \log(p(x)),$$

where $I(x) = \log(1/p(x)) = -\log(p(x))$ is the *self-information* of $x \in X$, and $E_{\mathbf{X}}(\cdot)$ the expectation function. Signals may be modeled as random variables, and therefore we can apply the functions previously defined. The *joint entropy* of two signals X, Y is computed as the entropy of the two paired signals:

$$H(X,Y) = E_{X,Y}(-\log(p(x,y))) = -\sum_{x \in X} p(x,y) \log(p(x,y)).$$

The *conditional entropy* of X given another random variable Y is defined as

$$\begin{aligned} H(X \mid Y) = E_Y(H(X \mid y)) &= -\sum_{x \in X, y \in Y} p(x,y) \log \frac{p(x,y)}{p(y)} \\ &= H(X,Y) - H(Y). \end{aligned}$$

Mutual information between two signals X and Y is

$$\begin{aligned} I(X;Y) &= -\sum_{x \in X, y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \\ &= H(X) - H(X \mid Y) = H(X) + H(Y) - H(X,Y), \end{aligned}$$

and it provides a measure of the amount of information about X that can be obtained by observing Y. Finally, the *Kullback-Leibler divergence* measures the difference between two distributions $p(X)$ and $q(X)$ for a signal X:

$$D_{KL}(p(X) \| q(X)) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}. \tag{2.8}$$

Another important concept is the *Kolmogorov complexity*, the measure of the shortest length of the description for a given string. It is undecidable, but it can be approximated using other techniques, such as lossless compression algorithms.

2.3 Bayesian Networks

We have so far provided the theoretical groundwork for developing the theory of Bayesian Networks; now we can define them, show how they work, and list some of their properties. A good starting point for the study of Bayesian Networks is Koller and Friedman [67]. We limit ourselves to the discrete case; however, a large portion of what follows also holds for the continuous case. We also almost completely skip any tractation of Markov Networks, the other large subfamily of Probabilistic Graphical Models. The reader may refer to [67] to deepen these subjects.

2.3.1 Definition

A *Bayesian Network* is a probabilistic graphical model encoding causal relationships among entities via a directed acycic graph; Bayesian Networks (in what follows, sometimes BNs or networks for short) have been proposed in Pearl [87]. Each node of the DAG is associated to an entity represented as a random variable, while the directed edges among the nodes encode a probabilistic causal relationship among the corresponding variables.

We begin to introduce the notation that we will use in the remainder of this thesis, in cases when no confusion may arise. Let $\mathsf{DAGS}(V)$ be the the set of possible directed acyclic graphs definable over a set of nodes V. A network $B \in \mathsf{DAGS}(V)$ is a directed acyclic graph $G = (V, E)$, where the set V of nodes represents a set of random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, with $n = |V|$, each X_i assuming a value taken from at most r_i possibilities. The terms *variable*, *node* and *feature* are synonyms in our context, although we will try to use each term in its more appropriate context, respectively probabilistic interpretation, graphical interpretation, and databases. The set of parent nodes for a node $v \in V$ will be denoted as s_v. A network can be represented by the set $S = \{s_1, \ldots, s_n\}$ of parent nodes of the variables, since this information contains all of the relationship among the variables.

The BN B for a dataset D is the network entailing the probability distribution that generated the data in D. Clearly, D has n features, that is, every feature tracked in the data has a corresponding variable X_i in the graph. We indicate the length $|D|$ of the database, that is,

the number of items in it, with N; N_{ijk} is the number of items in D where variable X_i takes its k-th value x_{ik} among r_i possible ones, and the variables in s_i take their j-th configuration w_{ij}, among $q_i = \prod_{X_j \in s_i} r_j$ possible configurations. $N_{ij} = \prod_{k=1}^{r_i} N_{ijk}$ is the number of instances in D there the variables in s_i take their j-th configuration; $N_{ik} = \prod_{j=1}^{q_i} N_{ijk}$ is instead the number of instances in D where X_i assumes its k-th possible value.

Each node i representing random variable X_i holds the conditional probability distribution $P(X_i \mid s_i)$. The probability of each configuration is computed as

$$\theta_{ijk} = P(X_i = x_{ik} \mid s_i = w_{ij}). \tag{2.9}$$

The whole network thus represents a valid joint distribution satisfying the chain rule

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid s_i). \tag{2.10}$$

We say that $P(X_1, \ldots, X_n)$ *factorizes* over G, and that G is an *independence map* (or *I-map*) for $P(X_1, \ldots, X_n)$. If $P(X_1, \ldots, X_n)$ factorizes over G, then G is an I-map for $P(X_1, \ldots, X_n)$; the viceversa does not always hold, since there may be independencies in $P(X_1, \ldots, X_n)$ that are not represented in G. We formally write $I(G) \subseteq I(P)$ for this, where $I(P) = \{(X, Y \mid Z) : P \models (X \perp\!\!\!\perp Y \mid Z)\}$ is the set of conditional independences that hold in $P = P(X_1, \ldots, X_n)$, and $I(G)$ is the set of d-separations in G. If $I(P) = I(G)$ than we have a *perfect map*.

Thus, a Bayesian Network B is fully defined by a triple $(\mathbf{X}, G, \Theta)$, where $\mathbf{X}$ is the set of features B is built over, the directed acyclic graph $G = (V, E)$ is the *structure* of the network and the set of probability distributions $\Theta = \{\theta_{ijk}\}$ contains the *parameters* of the network. The *skeleton* of a network is the graph obtained by removing the directionality from its edges, that is, the graph encoding only the connections among the nodes, without the causality notion.

Such a formulation is minimal, and can be extended if needed. For example, a Bayesian Network may have one or more *latent nodes* or *hidden nodes*, nodes in the graph that have no correspondent random variable or feature in the database, but allow to better encode the

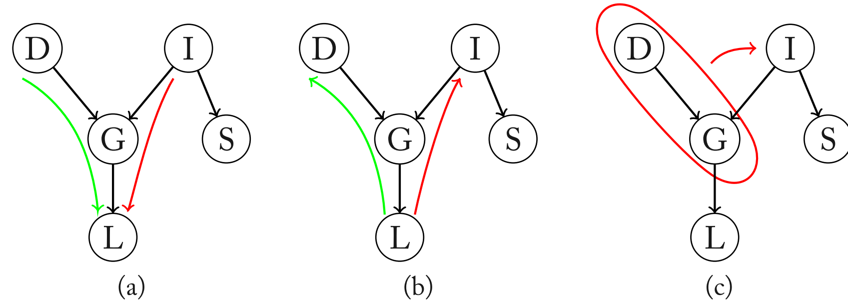


Figure 2.1: example of flows of influence in a Bayesian Network. Examples taken from `https://class.coursera.org/pgm-003/`.

independencies discovered in the database, corresponding to variables that have not been observed or measured. Bayesian Networks can also model evolving structures, for example along time sequences: in this case we have a *Dynamic Bayesian Network* (DBN). A DBN consists in a sequence of networks, each modeling a system in a different moment; nodes in networks of time t_i can have parents not only in the network of the same time slice, but also in the networks of the previous time slices $t_j, j < i$.

2.3.2 BNs and conditional independence

A Bayesian Network B over a set of random variables $\mathbf{X}$ and a DAG G represents the conditional independences among the variables in $\mathbf{X}$, and the set of those conditional independences forms the I-map $I(P)$. Therefore, a BN is a proper choice for analyzing the flows of probabilistic influence in G, that is, how each node can or cannot influence the other nodes in the graph. We introduce the explanation with the aid of the graphs pictured in figure 2.1.

The graph in figure 2.1 is a network modeling a simplified college graduation process: node D represents the difficulty of the course, I the intelligence of the student, G the final grade obtained by the student in the exam, S the mark of the student in the SAT test, and L is the event indicating that the teacher of the course provides a presentation letter for the student. Clearly, the difficulty of the exam and the intelligence of the student are intrinsic characteristics that do not depend on any other factor (at least in this toy example), and

therefore are *evidence variables* (we do know them); the result on the SAT test depends only on the abilities of the student, while the grade of the exam depends both on the student's capability and the course difficulty. Finally, a teacher is clearly more willing to provide a presentation letter to students who perform well in her course.

In figure 2.1a a *causal reasoning* pattern is represented: the lower the difficulty of the course, the higher the probability of getting a good grade, while ignoring the other factors involved (green arrow); at the same time, the more clever the student, the higher her probability of succeeding in the course (red arrow). Clearly, a high grade raises the probability of getting a cover letter from the teacher. Therefore, we see how D and I are both causes[1] of G and subsequently L.

Figure 2.1b shows instead the dual process, namely the *evidential reasoning* pattern. Here we *observe* the effects, and try to infer its causes. Being the SAT a standard test, a good score for it is a sign of a good student; a presentation letter suggests that the student performed well on the teacher's course; a good grade on a course may indicate that the student is clever (red arrow), but also that the course may be easy (green arrow). We say that S is an *evidence* for I, and L and subsequently G are evidence for both I and D.

Finally, 2.1c shows an example of *intercausal reasoning* pattern, also known as *explaining away*: if we observe some phenomenon with multiple possible causes, and already know that one of those possible causes happened, it reduces the probability of the remaining possible causes (some say that it *explains them away*), since already one event that can explain our observed phenomenon exists. In our toy network, for example, if a student has obtained a good grade in the exam, and we know that the course is difficult (and therefore the probability of obtaining a good score in it is low), we can infer that very likely the student is clever. Conversely, if a student has obtained a good grade, but it is a quite common event, we cannot deduce very much about

[1] In this example it may arise some confusion from the fact that *difficulty* and *intelligence* are contrasting factors for obtaining a high grade in the course; however, these are qualitatives considerations that are not represented in the network, and can be avoided by just using more general names for the parameters: for example, we can just compile the conditional probability table by assigning a higher probability to, say, the triple (g_1, d_1, i_2) with respect to the triple (g_2, d_2, i_1). This way, we have abstracted the network from its real counterpart (the examination process), and have therefore eliminated every source of confusion.

the abilities of the student, as we already know the course is easy. Expanding our example, if a student has a good mark on the SAT test and a low grade in the exam, likely the course is difficult.

We sum up the possible flows of conditional independence between two random variables X and Y, possibly whenever a third variable W is introduced; this works also for sets of variables. We suppose that no observation are made on the network. First of all, in a structure $X \rightarrow Y$, we have seen how both X can influence Y and Y can influence X, because of the two reasoning patterns that arise in this situation. Also in case of $X \rightarrow W \rightarrow Y$, X can influence Y (in fact, this is the chain rule), but also Y can influence X. If $X \leftarrow W \rightarrow Y$, that is, both X and Y are children of W, X can influence Y via W and viceversa. Instead, in the opposite structure $X \rightarrow W \leftarrow Y$, called *v-structure*, where W is a child of X and Y, X cannot influence Y, and viceversa.

Now we suppose to observe some variable $Z \neq X, Y$, and repeat our analysis of the possible structures. If $W \neq Z$ (or a set of nodes $\mathbf{W} \not\subseteq \mathbf{Z}$), the flows of probabilistic influence are the same. Instead, if also W is observed, the situation is reversed: while the flows of influence along paths from X to Y remain, the observation of a variable W in the path from X to Y blocks the flow of influence along the path; the same happens if W is a parent of both X and Y (note that there may be other paths between X and Y; however, we cannot say nothing about them, we only know that one path is blocked). Conversely, in a v-structure where X and Y are parents of W, the observation of W enables the flow of probabilistic inference among X and Y. Table 2.1 sums up the possibilities.

A path from X_1 to X_n is called an *active trail* given $\mathbf{Z}$ if for any *v-structure* $X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$, X_i or one of its descendants is in $\mathbf{Z}$, and no other X_j along the path is in $\mathbf{Z}$. If in G there is no active trail between X_1 and X_n given $\mathbf{Z}$, then we say that X_1 and X_n are *d-separated* in G given $\mathbf{Z}$ (we also write $\text{d-sep}_G(X, Y \mid Z)$, short for *directly separated*). Any node is d-separated from its non-descendants given its parents. If the parent nodes of X_i are linked by an edge, they are sometimes called *spouses*; conversely, if they are not directly connected, they form an *immorality*.

Table 2.1: summary of flow of influence along a path between X and Y: when can X influence Y given evidence about $\mathbf{Z}$?

	$W \notin \mathbf{Z}$	$W \in \mathbf{Z}$
$X \to Y$	✓	✓
$X \leftarrow Y$	✓	✓
$X \to W \to Y$	✓	
$X \leftarrow W \leftarrow Y$	✓	
$X \leftarrow W \to Y$	✓	
$X \to W \leftarrow Y$		✓

2.3.3 Properties

Two networks B_1, B_2 are *equivalent* if they encode the same set of conditional independences — that is, every joint probability defined by B_1 is also encoded by B_2, and viceversa.

Two BNs are *Markov equivalent* if they define the same structure; alternatively, they share the same set of adjacencies and immoralities. Subsequently, a *Markov Equivalence Class* is a set of networks that are Markov equivalent. The *essential graph* of a Markov Equivalence Class $\mathcal{M}$ is a graph that admits both directed and undirected edges: if a directed edge $(i \to j)$ is present in all of the graphs of $\mathcal{M}$ while no graph in $\mathcal{M}$ contains $(j \to i)$, then the essential graph for $\mathcal{M}$ also contains $(i \to j)$; otherwise, if both $(i \to j)$ and $(j \to i)$ are present in $\mathcal{M}$, then the essential graph for $\mathcal{M}$ contains the undirected edge (i, j). The essential graph is therefore a chain graph without flags. In figure 2.2c it is shown a simple example of essential graph for a network with three nodes, obtained from the two Markov equivalent networks 2.2a and 2.2b.

The *inclusion neighbourhood* (see Chickering [19]) is defined as follows: given $G \in \mathsf{DAGS}(V)$, let $\mathcal{I}(G)$ be the collection of conditional independences entailed by G. Let G, H be elements of $\mathsf{DAGS}(V)$ such that $\mathcal{I}(H) \subset \mathcal{I}(G)$ (note that $\mathcal{I}(H) \neq \mathcal{I}(G)$), and there is no $F \in \mathsf{DAGS}(V)$ such that $\mathcal{I}(H) \subset \mathcal{I}(F) \subset \mathcal{I}(G)$: then G, H are inclusion neighbours.

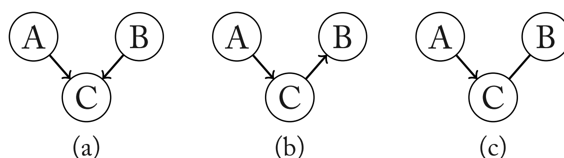


Figure 2.2: Example of equivalence classes for a simple graph: graph (c) is the essential graph for graphs (a), (b).

2.4 Mixed-Integer Linear Programming

An *optimization problem* is a problem where the desired outcome is the one that maximizes a certain function, called *objective function*, with respect to a set of *constraints*. For example, the Travelling Salesman Problem is an optimization problem where the quest is for the lowest-cost Hamiltonian cycle, modelled as the problem of an agent who has to visit a set of places and get back to the starting point, by the shortest path, without passing two times for a city (except for the starting point). *Linear programming* (LP) is a method for solving optimization problems in the form of a model composed by a linear objective function, and linear constraints. Linear programming lies in the P complexity class, since there are algorithms for it that run in polynomial time (Khachiyan [63], Karmarkar [59]). When the variables are requested to be integer, the problem is called an *integer linear programming* (ILP, or IP) problem. When the integrality is requested only on a subset of variables, it is said to be a *mixed-integer linear programming* (MILP, or MIP) problem.

Optimization problems with integer variables are ubiquitous in many fields, such as scheduling and planning in industry, or protein folding in biology, or many problems in graph theory. Unfortunately, integer programming is also a NP-complete problem: the special case of only binary variables is one of the Karp's 21 NP-complete problems (Karp [60]). Therefore we have scarce hope of finding an efficient algorithm to solve it. However, due to the importance of such problems, many algorithms, techniques, and software have been developed to tackle this problems in both exact and approximate way. In this section we briefly introduce the theory of integer linear programming, its geometrical interpretation, and some techniques used to solve it. Then, an overview of the available software is given, with

the presentation of some novel techniques called matheuristics.

For deeper and more complete references, several textbooks and works are available, e.g. Nemhauser and Wolsey [81], Schrijver [95], or Applegate, Bixby, Chvatal, and Cook [3] for a thorough introduction on theory applied on a practical problem.

2.4.1 Theoretical introduction

An integer linear programming problem is a problem in the form

$$\text{minimize } \mathbf{c}^{\mathbf{T}}\mathbf{x} \tag{2.11}$$

$$\text{subject to } \mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{2.12}$$

$$\mathbf{x} \in \mathbf{X}, \tag{2.13}$$

$$\mathbf{X} \subseteq \mathbb{Z}^n, \tag{2.14}$$

where $\mathbf{x}$ is the vector of variables in the set of admissible solutions $\mathbf{X}$, and $\mathbf{c}$ is the cost vector. The function 2.11 $\mathbf{c}^{\mathbf{T}}\mathbf{x} : \mathbf{X} \to \mathbb{R}$ is the objective function, while equations 2.12–2.14 are the constraints. $\mathbf{X}$ is the set of the feasible solutions; if no $\mathbf{x} \in \mathbf{X}$ satisfies the constraints ($\mathbf{X} = \emptyset$) the problem is infeasible; whenever, instead, the objective function has no global minimum for values in $\mathbf{X}$, the problem is unbounded.

Such a formulation for the problem is called the *canonical form*. If 2.12 is, instead, in the form $\mathbf{A}\mathbf{x} = \mathbf{b}$, the problem is said to be in *standard form*. The two formulation are equivalent, but the number of variables involved may vary. When, as in this case, the problem is to find the solution with the lowest cost, the problem is a *minimization problem*. Analoguously, a problem whose objective function has to be maximized with respect to the constraints is a *maximization problem*. It is possible to define an equivalent maximization problem for every minimization problem, and viceversa: such property is called the *duality in linear programming*.

Problems without the integrality constraint 2.14 become linear programming problems.

2.4.1.1 Geometric interpretation

The set of constraints 2.12–2.13 defines a convex polytope $\wp$ in a space of a number of dimensions as high as the number of variables involved

in the problem, that is, an intersection of hyperplanes and affine half-spaces for which the convexity property holds: for every $\mathbf{x}, \mathbf{y}$ in $\wp$, $\wp$ also contains all the points $\mathbf{z} = \lambda\mathbf{x} + (1-\lambda)\mathbf{x}, \forall\lambda \in (0,1)$ (all the convex combinations of $\mathbf{x}, \mathbf{y}$, excluding $\mathbf{x}$ and $\mathbf{y}$). Moreover, $\wp$ is bounded, that is, there is a value M such that $\|\mathbf{x}\| \leq M \ \forall \ \mathbf{x} \in \wp$. Every point in $\wp$ that cannot be defined as a convex combination of another two points ot the same polytope, is a *vertex* for $\wp$ (Minkowski-Weyl theorem).

Solving an LP problem is equivalent to evaluate the points of the polytope defined by the constraints imposed, in order to find the vertex (or one among the vertices) that minimizes (or maximizes, for maximization problems) the objective function. One fundamental consequence of the Minkowski-Weyl theorem is that we can restrict our quest to the vertices of the polytope, since no internal point can have lower (respectively: higher) objective value. The integrality constraint 2.14 requests the vertices of $\wp$ to have only integer coordinates.

2.4.2 Techniques for solving MILPs

2.4.2.1 Exact search

Due to the importance of the problem, and its difficulty, several strategies have been developed in order to solve linear problems, see Genova and Guliashki [49] for a survey. One possible idea is to move through the vertices of the polytope defined by the model, going from a candidate solution to another candidate solution with a better objective value. Probably the most important of such methods is the *simplex algorithm* (Dantzig [31]). Simplex algorithm has an exponential worst-case running time, but it works quite well in practice; in fact, it has been proved (Spielman and Teng [106]) that it has a quadratic running time on the average case. Simplex is easily implemented in the so-called *tableau* form, a table where the first row defines the objective function, and all the other rows specify the constraints; each variable is assigned to a column of the tableau. A similar approach is exploited in subgradient methods and coordinate descent algorithms, where the objective function moves along the coordinates of the polytope in order to find a local minimum. These methods, such as for example Lagrangian multipliers (see for example Hiriart-Urruty and Lemaréchal [55]), usually perform well in the first steps but, when get

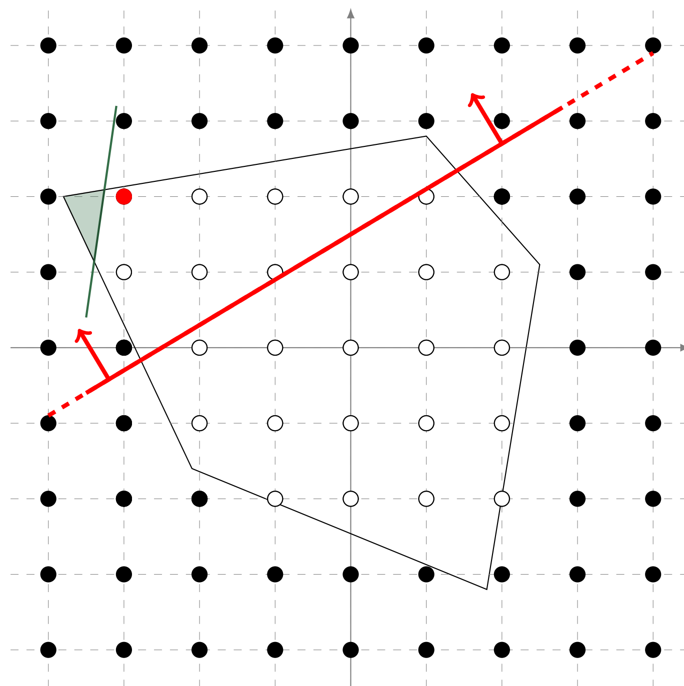


Figure 2.3: a polytope in $\mathbb{R}^2$: white points are the interior points of the polytope defined by the IP problem, while black points are points outside the polytope. The black perimeter is defined by the constraints over the LP relaxation. The red point is the optimal point, and the red line is the objective function, with its direction. In light blue, a cutting plane, and the section of the polytope it removes.

close to the optimum, end up moving around it; furthermore, their performances are very dependent on parameter tuning.

Another idea is to remove suboptimal vertices from the polytope: this is the *cutting planes* method (Gomory [52]). A cutting plane is just a new constraint added to the model, defining an hyperplane that intersects the polytope and narrows the search space. The risk of adding cutting planes is to prune also the optimal vertex, or to end up having mutually inconsistent constraints, so that the problem becomes infeasible. Furthermore, often there are a few constraints that shape the polytope, while all the rest of them are just redundant: finding such "good" constraints is crucial in the running time for the instance. However, being careful, cutting planes are a very powerful way of solving optimization problems: many cuts have been proposed

in the literature, like for example Gomory cuts ([52]) and Gomory-Chvátal cuts (Chvátal [22]). The problem of finding cutting planes is called *separation problem*.

When looking at the problem from its combinatorial point of view, several algorithmic paradigms are available, for different problem structures. If a problem is easily decomposable in smaller subproblems, one can use dynamic programming (Bellman [7], Leiserson, Rivest, Stein, and Cormen [72]). If the problem is a matroid, that is, its structure is composed of mutually independent sets, greedy algorithms can be used ([72]). Such approaches may lead to polynomial time algorithms. In other cases, when the matrix of constraints does not have some special structure and therefore we have to explore all the combinatorial space, the method of choice is perhaps the *branch-and-bound*, that constructs a tree from all the possible choices for each variable, and traverses it in depth-first order (Land and Doig [68]). Every node corresponds to a subproblem of the problem in the parent node, where a choice has been made, and there is therefore one less degree of freedom. The key advantage of this method is that, whenever after a choice we discover that such assignment is not optimal, the node of that choice, and all of its subtree, can be pruned, because the optimal solution will surely be somewhere else. Branch-and-bound is an implicit enumeration method, because, in the worst case, we have to check all the possible combinations of variables. Several authors used this method in late 50's in order to solve TSP instances; Land and Doig [68] showed how it applied to other problems too. The name is instead credited to Little, Murty, Sweeney, and Karel [74].

Branch-and-bound is a general paradigm that can be used in association with other methods: at every node, we can for example try to solve the corresponding subproblem with another algorithm, or even the simplex method. When, at every node, cutting planes are inserted, we call this the *branch-and-cut* method (name was given by Padberg and Rinaldi [86], but first implementation appeared in Hong [56]).

2.4.2.2 Approximate search

When an exact search in the combinatorial space is infeasible due to the size of the instances, one can choose to settle for a sufficiently

good solution, instead of the optimal one. Approximation algorithms are algorithms that try to guarantee a bound on the gap between the solution computed and the optimal one. Another large branch of non-exact methods is the family of *heuristic algorithms*, that is, algorithms aimed to quickly solve a problem, often without regard for the quality of the solution. Heuristics may be based on theoretical results, practical observations, or "rules of thumb" based on experience. Heuristics can be viewed, from a geometric perspective, as a *local search* in a restricted neighbourhood of a candidate solution. A popular class of heuristic algorithms is the class of *metaheuristics*, heuristics based on some metaphore (refer to Gendreau and Potvin [48] for a review, or Russell, Norvig, Canny, Malik, and Edwards [93]). Among them, we mention *tabu search* (Glover, Laguna, et al. [50]), *hill climbing* ([93]), *simulated annealing* (Kirkpatrick, Jr., and Vecchi [64]), , and biologically-inspired algorithms such as *ant-colony optimization* (Dorigo and Di Caro [39]), *particle swarm optimization* (Kennedy and Eberhart [62]) and *genetic algorithms* (Smith [101]). The main advantage of such methods lies in the their computational complexity, generally lower than complexity of exact algorithms. Their main drawback is, on the contrary, the inability of guaranteeing a worst-case bound on both the quality of the solution and the time needed to find it. To partially overcome the issues generated by the restriction of the search space and broaden the spectrum of candidate solutions, in many applications local search is run multiple times, from different starting position. Many approximation algorithms are also based on some restarting notion, often using randomization.

As we have already mentioned, when the problem is too difficult to solve with the given constraints, it is common practice to *relaxate* it, in order to solve an easier version of the problem Nemhauser and Wolsey [81]. Several methods are known to relaxate a problem. A common way of relaxate an IP is, as mentioned before, to remove the integrality constraints on some variables, or on all of them, obtaining, respectively, a MIP or a LP: this practice is known as *linear relaxation*. Another common approach is the *relaxation by elimination*, which removes some constraints to ease the search for a solution. The *surrogate relaxation* substitutes a set of constraints with a linear combination of them, in order to "weigh" them differently. *Lagrangean relaxation* removes a set of constraints, accounting for them in a modified objective function. Relaxations provide upper or lower bound on the value of

the optimal solution, but have to be handled carefully, since defining a relaxated problem basically means to define a *different* problem, so that optimal solutions for a relaxated problem may even be infeasible for the original problem. The geometric interpretation of a relaxation is an expansion of the original polytope into a bigger one, since the removal of some constraints results in the removal of some hyperplanes shaping the candidate solution space.

Relaxations are extremely useful not only for getting an approximate answer *per se*, but also for solving exactly huge instances that would otherwise be computationally infeasible. A common approach is to relaxate a problem removing constraints, solve it as usual, and then validate that solution also with the removed constraints, injecting them into the model as cutting planes. This method is also called *row generation*, because of its effect on the simplex algorithm in *tableau* form; a dual relaxation, the *column generation*, starts with a reduced set of variables, iteratively solving and adding variables as needed, until the optimum is found, or some stopping criteria are met. Refer to Nemhauser and Wolsey [81] for a more comprehensive tractation on the subject.

2.4.3 Software for solving MILPs

Problems in the form of integer programming can be solved using appropriate software, named *MIP solvers*. MIP solvers take in input a model and compute the optimal solution, together with its value. Many of these software exist nowadays, both commercial and free. Among the former group, the top products on the market are IBM ILOG CPLEX ™[2], Xpress ™[3], Gurobi ™[4]; as for the latter, COIN-OR[5] and GLPK[6] are among the most widespread choices.

MIP solvers are built to solve mixed-integer linear programming problems. Therefore they can solve every optimization problem, provided it is in MIP form. These solvers essentially implement a branch-and-cut strategy: at every node they apply a portfolio of heuristics and

[2] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[3] http://www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx

[4] http://www.gurobi.com/

[5] http://www.coin-or.org/

[6] http://www.gnu.org/software/glpk/

general-purpose cuts, and try to solve the problem; then, if the problem has not been solved after such processing, the solver chooses a variable and branches on it. In addition to this, solvers usually implement many of the techniques listed in section 2.4.2, and carry the result of years of parameter-tuning in order to accomodate the majority of the customers out of the box. Futhermore, commercial solvers also implement proprietary solutions, such as the infamous CPLEX dynamic search, an algorithm to determine how to develop and traverse the branching tree.

Solvers provide several methods and libraries to embed them in other software, from the simple creation of a model, to callbacks that intervene in various occasions during the execution of the solver, for example when a feasible solution is found. Commercial software, however, may disable some of their functionalities in this case, in order to prevent the users to reconstruct their proprietary mechanisms from the behaviour of the solver when manipulated in a certain way. Therefore, the developer who wants to interact in such close way with the solver has to be careful in which actions he or she undertakes. Nonetheless, new algorithms based on this possibility have been proposed, and we describe them in section 2.4.3.1.

Another widespread non-commercial software is the framework SCIP[7] (Achterberg [1], Berthold et al. [8]) from the ZIB institute in Berlin, that implements a branch-and-cut and a variety of solvers, heuristics, cuts, letting the user the choice of which parts to use, how, and when. SCIP is not open source, but it provides an interface to its internal methods, so that programming with SCIP essentially amounts to configure it enabling a series of parameters. In figure 2.4 we report a chart comparing the geometric mean of the time needed by some commercial and non-commercial solvers with respect to SCIP, as of January 2013. We note how the industrial solutions perform drastically better then their free counterparts, as a consequence of the efforts put in the race for the best product of the market.

[7] http://scip.zib.de/scip.shtml

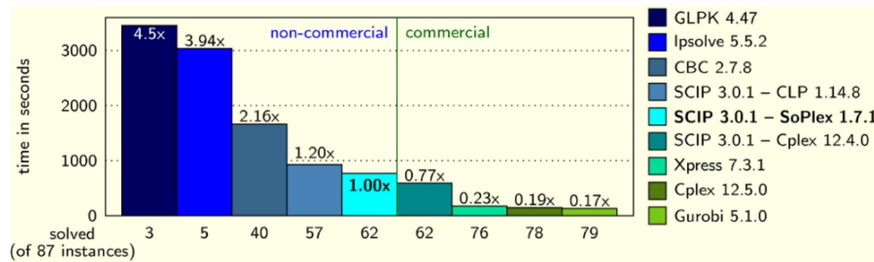


Figure 2.4: comparison of MIP solvers with respect to SCIP (source: `http://scip.zib.de/scip.shtml`)

2.4.3.1 Solvers as heuristics for binary problems

Over the years, industries in the optimization business branch have built increasingly powerful pieces of software that can be used to solve an enourmous variety of problems, in less and less time. Such solvers are extremely optimized yet general in their purpose, and constantly better year after year, so that the state-of-art solvers of today are much better than the state-of-art solvers of the past year, thanks to both new theoretical solutions and new tweaks and tuning. Thus, in recent years it has sprung the idea of using them in a "black-box" fashion to quickly obtain good solutions for big problems, instead of waiting for an optimal solution that can take a long time to find or to certify. This concept of embedding a MIP solver as a step of another algorithm has been given the name of *matheuristics*, a portmanteau for *heuristics based on mathematical programming* that also resembles the more widespread term *metaheuristic* (see Maniezzo, Stützle, and Voß [78]).

From an algorithmic point of view, matheuristics essentially perform a local search in the geometric neighbourhood of an initial solution, just like more traditional heuristics such as biologically inspired metaheuristics, or tabu search. Initial solution that can be provided by the user, or computed by the MIP solver itself, starting from a linear programming model that can be conveniently relaxated if needed. Therefore, it is possible to get back a solution from scratch, or to have a starting solution refined. Matheuristics are designed to obtain a quick, good enough solution; as a local search paradigm, matheuristics suffer from the drawbacks of their more traditional siblings, mainly being stuck in local optima, and impredictable, erratic behaviour (Lodi [76]).

Algorithm 2.1: MATHEURISTIC GENERAL FRAMEWORK.

```
   Data: problem
   Result: approximate solution for the problem
 1 create the (relaxated) model for the problem;
 2 if starting solution is provided then
 3 |  pass it to the solver;
 4 end
 5 while stopping criteria not met do
 6 |  solve problem using a MIP solver;
 7 |  if some constraints are violated then
 8 |  |  add them to the model;
 9 |  else
10 |  |  stop;
11 |  end
12 end
13 return solution;
```

From a practical point of view, matheuristics have been proved very convenient to use, since they can be applied to virtually every problem, and have the merit of using some extremely performing software as a subroutine, instead of coming up with an efficient algorithm that of course needs to be coded and tuned correctly and efficiently. Many times, implementing a matheuristic reduces to code the steps in algorithm 2.1. Moreover, since solvers already have many heuristics in them, it is possible to have lower and upper bounds on the value of the solution, and therefore a measure, perhaps rough, of the quality of the solution we are going to get back. Stopping criteria can be, for example, a time limit, or the quality of the solution.

Anyway, such methods may even fail to find a feasible solution, so that one has to go back to other ad-hoc heuristics.

Among the matheuristics that have been proposed in the last years, we mention the *hard fixing* of variables (Bixby, Fenelon, Gu, Rothberg, and Wunderling [11]), a method that fixes some of the variables before solving the problem, that after such preprocessing will be easier, since assigning a value to a variable means to lower the degrees of freedom. This is a method also called *diving*, because its effect in terms of branching tree is to "jump down" into lower levels, just solv-

ing one subtree while ignoring all of the remaining possible ones. A conceptually similar idea is proposed by Fischetti and Lodi [40] with *Local Branching*, that iteratively explores limited subtrees, partitioning the solution space into narrower areas using constraints. *Proximity Search* is a novel matheuristic introduced in Fischetti and Monaci [42], that limits the search in the neighbourhood of a given feasible solution $\tilde{\mathbf{x}}$. This is basically achieved introducing a *cutoff value* θ, adding a *cutoff constraint* $\mathbf{c}^\mathbf{T}\mathbf{x} \leq \mathbf{c}^\mathbf{T}\tilde{\mathbf{x}} - \theta$ whose purpose is to "drive" the objective function through the search space, and replacing the original objective function $\mathbf{c}^\mathbf{T}\mathbf{x}$ with the proximity objective function $\Delta(\mathbf{x}, \tilde{\mathbf{x}})$. When a feasible solution for the modified problem has been found, the distance function is recentered on this newfound solution, and the search is repeated on the redefined search space. Such procedure is iterated until stopping criteria are met.

CPLEX implements an approach called *RINS*+Polishing, based on the *Relaxation Induced Neighborhood Search* proposed in Danna, Rothberg, and Le Pape [30] and the polishing approach of Rothberg [92]. Given an incumbent solution $\tilde{\mathbf{x}}$ and the optimal solution of the LP relaxation at the root node $\mathbf{x}^*$, RINS looks for the variables that "agree" in both solutions: such variable values (either 0 or 1) will likely be the optimal ones. Then, the algorithms tries to "link " the two solutions, moving from a solution to the other one; this is the *path relinking* from Glover, Laguna, and Martí [51]. When a good feasible solution is found, the *polishing* step "refines" the solution, exploring the neighbourhood to check whether there are better solutions near the one found. When the polishing is launched on a set of feasible solutions, this approach takes the name of *MIP-and-refine*.

The aforementioned approaches are targeted to binary mixed-integer LPs, but can be applied to general MILPs for example by encoding non-binary variables in binary form, see e.g. Rossi, Petrie, and Dhar [91], Dechter and Pearl [38]. Deeper and more complete surveys of matheuristics can be found in Maniezzo, Stützle, and Voß [78], Fischetti, Lodi, and Salvagnin [44], Fischetti and Lodi [41], Lodi [76]. However, due to the usefulness and versatility of MIP solvers, not to mention their increasing power, many other approaches have been proposed, and new ones are expected to be proposed.

3 Bayesian Network Structure Learning

One of the key problems when dealing with Bayesian Networks is how to discover the original network from a given database, or at least the most probable network given the observed data. This issue can be divided in two parts: first, to discover the structure of the network, the graph that connects the variables, then estimate the parameters of the network. We will focus on the first part of the task, the *Structure Learning* problem.

The problem of learning the exact structure of a Bayesian Network from a a set of data is known to be NP-hard (Chickering [17]). Moreover, even learning an approximate structure is NP-hard, and so remains also when constraints on the maximum number of parents are added, besides the trivial case of a maximum of one parent per node, the problem becoming to find the optimal tree (Chickering, Heckerman, and Meek [21], Dasgupta [32]). The main difficulty lies in the acyclicity constraint, which imposes dependency among the parent sets for the nodes.

Several approaches have been proposed in the literature both for exact and approximate search, usually based on *scoring* the candidate solutions. Indepedence tests have been also proposed (see Spirtes, Glymour, and Scheines [107]) as an alternative to score-based methods, but, in general, real world problems do not satisfy the assump-

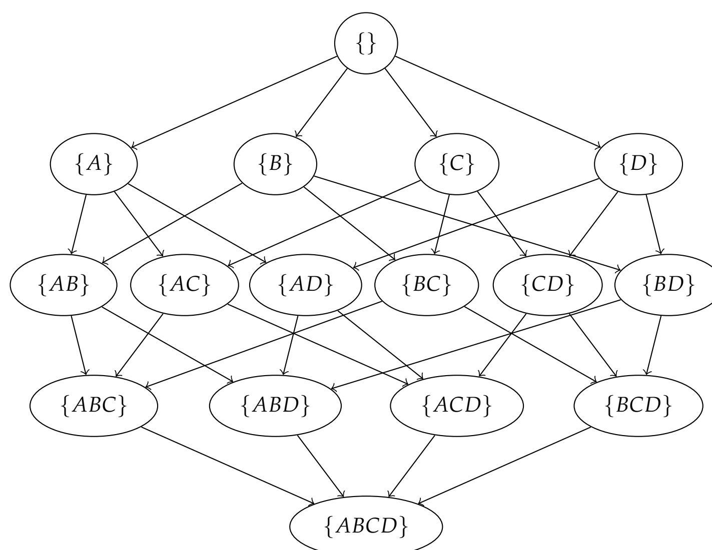


Figure 3.1: parent set lattice for 4 variables.

tions of independence assumed by this approach, so few algorithms aim to learn independence constraints from data. A notable exception is the Max-min Hill-climbing presented in section 3.3.3, and some derived works. Different approaches also result in different formal definitions of the problem, thus yielding different insights on it.

In this chapter we will present some of the most popular methods for learning a network from data; before this, however, a review of score metrics is given, because they provide the measure of the quality of the solutions, which is what guides the algorithms through the search space.

3.1 Scoring metrics

Because of the vast majority of learning methods are based on some scoring functions, in this section we review the most used scores proposed in literature. Scoring functions measure how likely each configuration parents-children ($s_v \to v$) is. This is a heavy task, because the number of candidate parent sets for each node is exponential; the candidate parent sets can be represented as a lattice as the one in figure 3.1, where each set of variables is formed by the union of its subsets.

A *scoring function* is a function $\text{Score}_D(s_v \to v) \mapsto \mathbb{R}$ whose outcome is a measure related to the probability of the substructure $(s_v \to v)$ to be in the network, according to the dataset D.

The problem of learning a Bayesian Network using a score-based approach, can be defined as the optimization problem of discovering the network that maximizes the overall score for the network, formally

$$\text{maximize } \text{Score}(G) \tag{3.1}$$
$$\text{s.t. } G \in \text{DAGS}(V) \tag{3.2}$$

where V are the nodes corresponding to the variables $X_1, \dots, X_n$ measured in the dataset, and $\text{DAGS}(V)$ is the space of DAGs over V.

3.1.1 General properties

When it comes to choose the score for an algorithm, we would like for it to have some properties, to ensure they respect the aforementioned definitions and properties of Bayesian Networks.

A scoring criterion is *decomposable* if the score of a DAG is the sum of local scores of subsets consisting in a node and its parents.

A scoring criterion is *score equivalent* if two Markov equivalent DAGs have the same score.

We also would like for the score to be *consistent*, that is, if the database D comes from m independent and identically distributes samples, as m grows, it holds that: *(a)* if node $p \in A$, $p \notin B$, $\text{Score}(A \mid D) > \text{Score}(B \mid D)$, and *(b)* $p \in A, B$, A has fewer parameters than B, $\text{Score}(A \mid D) > \text{Score}(B \mid D)$.

Another interesting property is the *local consistency of score*, meaning that, if A id a DAG, and B results from adding the arc $(u \to v)$ to A, as the size of database m grows, the score of B is greater than the score of A if the insertion $(u \to v)$ rules out an independence constraints that does not hold in D, and is lower if such insertion does not rule out any independence constraint that does not hold in D. In practice, only adding useful edges increases the score, while adding useless complexity lowers the score.

Consistency of score entails a superset pruning property, that allows us to discard a candidate parent set $V' \supset V$ if

3.1.2 Bayesian scoring functions

Bayesian scorings are a family of scoring functions that aim to maximize the posterior probability distribution from a prior probability distribution of the possible networks definable over a graph G, conditioned on the dataset D. A higher likelihood function denotes a structure more tailored to the data, and therefore more likely to be somewhat correct.

This family of scores is based on the *Bayesian Dirichlet* score (BD, Heckerman, Geiger, and Chickering [53]), based on some assumptions:

multinomial sample for each node in the network, its parameters depend only on the state of its parents;

parameter independence parameters in each variable in the structure are independent from parameters in other variables (global p.i.); moreover, parameters associated to each state of the variable, are independent from each other (local p.i.);

parameter modularity for each node, the densities of its parameters depend only on the node and its parents;

Dirichlet distribution node parameters have *Dirichlet distribution*; formally, let B_S be a network structure with non-zero probability given the data, and let B_S^h the hypothesis that the dataset D is generated by an unknown network structure B_S; then, there are exponents N'_{ijk} that depend on B_S^h and the current state of information ξ satisfying $\rho(\Theta_{ij}|B_S^h, \xi) = c \cdot \prod_k \theta_{ijk}^{N'_{ijk}-1}$, where c is a normalization constant;

complete data the database has no missing data, or missing data have been imputed.

Such set of properties form a sufficient condition to represent a user's prior knowledge, enabling the user to compute the probability of new cases given the data, the structure, and the current state of information, for every database and structure. From this set of assumptions, the authors define the posterior probability of the network given the

data:

$$P(G \mid D) = P(G) \times \prod_{i=1}^{|V|} \prod_{j=1}^{q_i} \left(\frac{\Gamma(N'_{ij})}{\Gamma(N_{ij} + N'_{ij})} \times \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk} + N'_{ijk})}{\Gamma(N'_{ijk})} \right), \tag{3.3}$$

where N_{ijk} are counter variables, N'_{ijk} are the hyperparameters, $\Gamma(\cdot)$ is the *Gamma function* $\Gamma(x+1) = x!$, and $P(G)$ is the prior probability for graph G. Since $P(G)$ is, generally, the same for all the possible DAGs, it is just a rescaling factor, and can be omitted. Due to computational considerations, the BD score is the logarithm of formula 3.3:

$$\begin{aligned} \mathrm{BD}(G \mid D) = \ & \log(P(G)) \sum_{i=1}^{|V|} \sum_{j=1}^{q_i} \Bigg(\log \frac{\Gamma(N'_{ij})}{\Gamma(N_{ij} + N'_{ij})} + \\ & \sum_{k=1}^{r_i} \log \frac{\Gamma(N_{ijk} + N'_{ijk})}{\Gamma(N'_{ijk})} \Bigg). \end{aligned} \tag{3.4}$$

In practice, the BD score is infeasible, since it needs to specify all the hyperparameters. One possible solution is to set all the hyperparameters to 1, thus falling into the case of the *K2* score (Cooper and Herskovits [24]):

$$\begin{aligned} \mathrm{K2}(G \mid D) = \ & \log(P(G)) \sum_{i=1}^{|V|} \sum_{j=1}^{q_i} \Bigg(\log \left(\frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \right) + \\ & \sum_{k=1}^{r_i} \log(N_{ijk}!) \Bigg). \end{aligned} \tag{3.5}$$

However, under the following further assumptions, two derived scores can be defined.

likelihood equivalence any two equivalent (given the data) network structures, share the same parameter distribution;

structure possibility every complete network structure has non-zero probability, given the data.

Assumptions 1–7 imply:

$$P(G \mid D) = P(G) \times \prod_{i=1}^{|V|} \prod_{j=1}^{q_i} \left(\frac{\Gamma(N'_{ij})}{\Gamma(N_{ij} + N'_{ij})} \times \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk} + N'_{ijk})}{\Gamma(N'_{ijk})} \right), \tag{3.6}$$

where N'_{ijk} is given by $N' \times P(X_i = ik, s_{X_i} = w_{jk} \mid G)$, for N' being the *equivalent sample size* representing the strength of the user belief in the prior distribution. Again, in practice it is computed in logarithmic form, and called the *Bayesian Dirichlet equivalent* score (BDe, Heckerman, Geiger, and Chickering [53]):

$$\begin{aligned} \text{BDe}(G \mid D) = \; & \log(P(G)) \sum_{i=1}^{|V|} \sum_{j=1}^{q_i} \Bigg(\log \frac{\Gamma(N'_{ij})}{\Gamma(N_{ij} + N'_{ij})} + \\ & \sum_{k=1}^{r_i} \log \frac{\Gamma(N_{ijk} + N'_{ijk})}{\Gamma(N'_{ijk})} \Bigg), \end{aligned} \tag{3.7}$$

$N' \times P(X_i = ik, s_{X_i} = w_{jk} \mid G)$. Also for BDe, defining all of the hyperparameters may not be easy. A special case is the score proposed in an earlier work by Buntine [15], called the *Bayesian Dirichlet equivalent uniform* (BDeu) score, who assumes the joint distribution of the states to be uniform: $P(X_i = ik, s_{X_i} = w_{jk} \mid G) = \frac{1}{r_i q_i}$. Therefore, we can write

$$\begin{aligned} \text{BDeu}(G \mid D) = \; & \log(P(G)) \sum_{i=1}^{|V|} \sum_{j=1}^{q_i} \Bigg(\log \frac{\Gamma(N'_{ij}/q_i)}{\Gamma(N_{ij} + N'_{ij}/q_i)} + \\ & \sum_{k=1}^{r_i} \log \frac{\Gamma(N_{ijk} + N'_{ijk}/r_i q_i)}{\Gamma(N'_{ijk}/r_i q_i)} \Bigg). \end{aligned} \tag{3.8}$$

3.1.3 Information theoretic scoring functions

The general idea behind such approach is that a Bayesian Network for some dataset can be thought as the generator of a code that can be used to compress the data. We can subsequently apply the concepts of information theory such as entropy and mutual information, as well as Shannon limit, in order to characterize the scoring methods for

Bayesian Networks. The lenght of describing the data D according to an hypothesis H can be thought as

$$L(D \mid H) + L(H), \tag{3.9}$$

that is the length of the description of the data when coded using the hypothesis, plus the length of the description H itself. In our case, the hypothesis H is instead a DAG G that "fills in" the database according to the probability distributions and set of conditional independences it represents.

A first scoring method is the *log-likelihood score*, computed as

$$\text{LL}(G \mid D) = \sum_{i=1}^{|V|} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}}. \tag{3.10}$$

Comparing it to the general structure given in formula 3.9, it measures only the first length, while ignoring the complexity of the hypothesis. Thus, the log-likelihood has some important limitations: it rewards complex structures, and it does not represents the conditional independences of the network. In learning theoric terms this leads to the *overfitting* issue, the risk of tailoring the solution to the training data, resulting in a wrong model that fails to perform correctly over arbitrary data. In order to reward simpler hypothesis, new scoring measures have been proposed, by adding a complexity penalty to the log-likelihood of the hypothesis. Such penalty is, basically, the difference between the following methods.

The *Akaike Information Criterion* (AIC, Akaike [2]), the *Bayesian Information Criterion* (BIC, Schwarz [96]), and the *Minimum Description Length* (MDL, Rissanen [89]) are three similar scoring methods (in fact, BIC coincides with MDL). They adopt a formalization of the *Ockham's razor*, which asserts that, among a set of admissible hypothesis for some phenomenon, the easiest one is the more likely to be the correct one. Formally, such methods have the form

$$\text{Score}(G \mid D) = LL(G \mid D) - f(|D|)|B| \tag{3.11}$$

where

$$|B| = \sum_{i=1}^{|V|} (r_i - 1) q_i \tag{3.12}$$

is the *network complexity* that depends on the number of variables, their number of possible states, and the number of possible configurations of their parents, and the penalization function

$$f(|D|) = \begin{cases} 1 & \text{for BIC/MDL} \\ \frac{1}{2}\log(|D|) & \text{for AIC.} \end{cases} \tag{3.13}$$

The introduced complexity penalty provides a very useful property to limit the search space: optimal parent sets have few nodes; this is in fact where the Ockham's razor shows up. Using these scores, the size for candidate parent set can be limited to $O(\log|V|)$ nodes. More precisely, this bound is $\log|V|$ for AIC/BIC (de Campos and Ji [34]), and $\log\frac{2|V|}{\log|V|}$ for MDL (Tian [118]). The superset pruning property also holds.

Another information theoretic based scoring criterion is the *Mutual Information Tests* (MIT, De Campos [35]). MIT measures the mutual information (see section 2.2.2) between a variable v and a candidate parent set s_v for it; of course, the candidate parent set which maximizes the score is the one to be selected. Formally,

$$\mathrm{MIT}(G \mid D) = \sum_{\substack{v=1,\\ s_v \neq \emptyset}}^{|V|} \left(2 \cdot |D| \cdot I(v; s_v) - \sum_{j=1}^{|s_v|} \chi_{\alpha, l_{ij}} \right), \tag{3.14}$$

where the innermost summation is a penalization term taking into account both network complexity and its reliability, being related to the Pearson χ^2 test of independence.

A more recent scoring function is the *Normalized Maximum Likelihood* (NML, Silander, Roos, Kontkanen, and Myllymäki [99]), which exploits the information theoretic interpretation of the MDL score: it aims to select the hypothesis with the smallest description that can generate, hence explain, the data. The most "natural" approach would be the Kolmogorov complexity, which is undecidable; therefore, a penalization term is added to the usual log-likelihood, trying to address some of the limitations of a simple penalty based on computing the complexity of a network. First, as parameters of the network are probabilities distributions, if some of these probabilities are 0, they do not need to be considered; secondly, if the same probability distribution, or the same probability values, occur multiple times in the network,

some pattern can be exploited to achieve a better compression. The solution is based on the idea of *universal coding*, choosing the best coding (the code with lowest length) for the data, *before* observing the data, that corresponds to finding the hypothesis for the data with the highest likelihood. In general, there is no such code, and so we have to design a code to compress the data in a way that we expect to be as closest as possible to the desired one. The corresponding hypothesis is called the *best-fitting hypothesis*; we can evaluate the performance of a set of hypothesis $\mathcal{H}$ for a distribution $\hat{H}$ by computing the *regret* of $\hat{H}$ relative to $\mathcal{H}$ for data D as

$$-\log(P(D \mid \hat{H})) - \min_{H \in \mathcal{H}}(-\log(P(D \mid H))), \tag{3.15}$$

which can be rewritten as

$$-\log(P(D \mid \hat{H})) + \log(P(D \mid H_{\mathcal{H}}(D))) \tag{3.16}$$

for the hypothesis $H_{\mathcal{H}}(D) \in \mathcal{H}$ that minimizes the second term; the *worst-case regret* for all of the hypothesis over the data of the same size $|D|$ is given by the maximum possible regret 3.16 over the data for a given size $|D|$. The *universal distribution* $H_{\mathcal{H}}(|D|)$ relative to $\mathcal{H}$ is the hypothesis $\hat{H}$ that gives the minimum worst-case regret over all the distributions on the data of size $|D|$. Such universal distribution is also called the *normalized maximum likelihood distribution*.

3.2 Independence tests

As already said, the alternative to scoring-based methods for learning networks is to perform independence tests among the variables (see 2.2). The purpose of these tests is to conjecture an independence hypothesis among a set of variables, and determine whether the hypothesis has to be accepted or rejected. Such default hypothesis is called the *null hypothesis*; independence tests aim to collect sufficient confidence in order to reject it, or, instead, to affirm that our data just happens to have values resembling “something different” just by random chance.

3.3 Algorithms

A plethora of algorithms have been proposed to solve the problem of learning the strcture of a Bayesian Network from data, following many different strategies and ideas. Many of them are designed to solve the problem in an exact manner; however, due to the computational issues already examinated, they are often imposed some limitation, in order to trade optimality with time performances. The most common relaxation of this kind is to impose a limitation on the number of maximum parents allowable for each node; others include to look for a slightly different network, easier to compute, to assume a known ordering of the nodes, or to terminate the algorithm at a certain point, for example after a given amount of time. Another approach is to run a local search from a starting point, being it a feasible DAG, a DAG that violates some constraints, or an undirected graph. Also, a way to speed up the computation is to subsample the dataset from which the network is generated.

For the current state-of-art, optimal reconstruction is possible for networks with no more that 20–30 nodes; approximate solutions can instead be computed for much larger networks, depending on the degradation over the quality of the solution we can tolerate. Usually the algorithms assume complete data. If the starting dataset has missing data, in order to employ one of the strategies we're presenting here one has to guess the missing items.

An algorithm that, given an infinite amount of data, is theoretically able to reconstruct the optimal network, is called a *sound algorithm*. The soundness property replaces the correctness notion in the stochastic domain, since the correctness of the outcome depends on the input.

In case we accept a premature termination we may be interested in a so called *anytime algorithm*, an algorithm that in any moment has a feasible solution, even if suboptimal, to return if asked to.

In the rest of this chapter, we briefly introduce some among the most popular or the most recent works proposed in the literature.

3.3.1 Dynamic Programming

Dynamic programming (see 2.4.2.1) is an algorithmic paradigm that has been applied by several authors to the problem of structure learn-

ing. These approaches rely on the decomposability property of scoring functions such as BD, AIC, BIC, in order to build up a table containing all of the possible combinations of subnetworks. This approach clearly leads to exponentially big tables, and therefore to exponential time algorithm, but the since they explore all of the possible combinations, they ensure to discover the optimal network.

Koivisto and Sood [66] propose an algorithm to evaluate the posterior probability of any edge (or every given subnetwork) in $O(n^3 2^n)$, based on computing the posterior marginal edge probability. Later, Koivisto [65] lowers this complexity down to $O(n2^n)$ by simultaneously computing the posterior probability for every edge.

Independently, about at the same time an analoguous solution has been proposed by Ott, Imoto, and Miyano [85], who apply this method to the discovery of gene networks.

Silander and Myllymäki [98] note that every DAG must have at least one leaf, a node without children. They call these nodes *sinks* of the network. Starting from this simple observation, they develop an algorithm that looks for an ordering of the variables in the DAG: in an ordering $ord = (v_1, v_2, \ldots, v_n)$, there may be an arc $(v_i \rightarrow v_j)$ only if $i < j$. A Bayesian Network is consistent with some variable ordering ord if, for every node i, its parent set s_i is made of nodes that precede i in ord, or, formally, $s_i \subseteq \bigcup_{j=1}^{i-1} \{ord_j\}$, where ord_j is the j-th variable in order ord. The algorithm is conceptually very simple: first, all the possible scores for each couple (node, candidate parent set) are computed. From this, the best parent sets and the best ordering are computed; combining these informations, it is possible to reconstruct the exact network.

Another dynamic programming approach, published in the same time of the previous work, is OptOrd of Singh and Moore [100]. It is based on the same idea of finding leaves developed in Silander and Myllymäki [98], but providing a theorem in order to avoid unnecessary computations.

3.3.2 Greedy

Greedy algorithms (see 2.4.2.1) have also been proposed. They start from an initial solution (which may be with or without edges) and iteratively apply one among a set of operations, looking for the choice

that (locally) maximizes the score improve. The set of possible operations consists of edge insertion, edge removal, and edge reversal (switching the directionality).

Meek [80] introduces the *Greedy Equivalence Search* (GES) greedy algorithm based on a conjecture regarding the transformations between DAGs in the same equivalence class. Informally, the *Meek conjecture* can be stated as follows.

Proposition 3.3.1 *For any two DAGs G, H, if H is an independence map for G, there is a finite sequence of edge operations (addition, removal, reversal) such that after every operation H is still an independence map for G, and, after all the operations, $H = G$.*

Assuming this conjecture was true, Meek proposed a two-step greedy algorithm, in which, given an equivalence class with no dependencies, first the dependencies are imposed by edge addition until a local maximum is found (according to some scoring criterion), and then edges that can be removed in all the DAGs in the equivalence class are deleted.

Conjecture 3.3.1 has been later proved to be true (Chickering and Meek [20], Chickering [18, 19]). The authors have been able to limit the search space for the edges, thus improving the performances of GES.

3.3.3 Max-min hill-climbing

Tsamardinos, Brown, and Aliferis [120] provide a double step algorithm, called *Max-min hill-climbing*, or MMHC for short, which first executes a statistical conditional independence test in order to find a reduced set of candidate parent sets, and then applies a greedy hill-climbing step to look for a (possibly locally) optimal solution. It is a particular case of the algorithm from Friedman, Nachman, and Peér [45], the first structure learning algorithm to be successfully applied to instances with hundreds of nodes. Authors classify their algorithm as a hybrid method between scoring-based methods and approached based on statistical independence tests.

The “max-min” step, based on the *Max-min parent-children*, or MMPC, is an heuristic that looks for a set of candidate parent set for each node in the graph. It aims to find an undirected graph representing the skeleton of the original DAG by looking for subsets of

variables $\mathbf{Z}$ conditionally separating pairs of variables X, Y. Such test is denoted as $T(X, Y \mid \mathbf{Z})$, and is the G^2 test (see section 2.2).

MMPC works as follows. Let PC_v^G be the set of parents and children for node v in a Bayesian network G over a probability distribution P; if G' is a Bayesian Network over P that is Markov-equivalent with respect to G, it holds that $PC_v^G = PC_v^{G'}$, and we can write only PC_v. Thus, the set of parents and children of a node is the same for all the Markov-equivalent Bayesian Networks over the same probability distribution P. MMPC performs the conditional independence test $T(X, Y \mid \mathbf{Z})$ in a subroutine called MinAssoc. The logarithm of the so obtained p-value is called the *association between variables*.

The Max-min heuristic, for every variable X, iteratively constructs a set of variables with high association with X (the CPC set), choosing at each iteration the variable Y with the largest association, until such association falls under a minimum value. As this may include false positives, variables that are not mutually included in respective CPCs are pruned. The significance of this test depends on the size of the database; since this also impacts the computational time in an opposite manner, a later work of Tsamardinos and Borboudakis [119] suggests to apply permutation tests in order to have significative tests also with limited datasets. The set of CPCs found by the Max-min heuristic form the skeleton of the Bayesian Network.

The hill-climbing algorithm is then applied to the skeleton in order to reconstruct the directionality of edges. This algorithm performs a local search, by applying the three possible operations over DAGs (edge insertion, edge removal, edge reversal) and greedily choosing the operation that increases the score the most until no improvement is found, relying on a tabu list to avoid cycles. The search space for edges is limited to the ones allowed by the CPCs found in the Max-min step.

Authors claim how their algorithms can address some common issues of network learning strategies, namely, the (un)soundness of learning, that is, reconstructed network does not represent the real distribution, and issues related to the enforcement of a maximum cardinality for candidate parent sets for the variables.

3.3.4 Branch-and-bound

De Campos, Zeng, and Ji [33] introduce an exact, anytime approach

based on constraints (already developed in sections 3.1.2, 3.1.3), subsequently improved in de Campos and Ji [34]. Such approach relaxates the problem by removing the acyclicity constraints, and then looks for the solution with a branch-and-bound strategy. The algorithm maintains a priority queue of candidate solutions ordered by decreasing scores, a "cache" for local scores, a matrix specifying whether arc is allowed, mandated or forbidden, an upper bound and a lower bound on the optimal score, and some control parameters. Having removed the acyclicity constraint, candidate solutions in the queue are (initially) likely to contain cycles.

The cache of local scores can be pruned by applying the constraints on the bounds. Control parameters are used to choose whether a graph to be analyzed should be taken from the top of the priority queue, or from the bottom. A graph taken from the top of the queue is possibly the optimal DAG, or it is used to enhance the upper bound; a graph taken from the bottom of the queue is used to raise the lower bound. If the graph processed has a score that is worse than the optimal score that the search has found so far, it can be discarded (the bounding step). If the graph is taken from the top of the queue, is a DAG, and has a better score than the current best solution, than it is the optimum, and the algorithm can stop. If the graph is taken from the top and has a directed cycle, such cycle is broken and new disjoint graphs are created (the branching step), provided the scores of these new graphs do no fall outside the bounds.

With such reiterated removal and inserting of graphs in the queue, elements from top tend to be high-scoring graphs with cycles, elements from bottom tend instead to be simpler DAGs.

Branch-and-bound with constraints is reported to work well also with a higher number of variables; tests reported by the authors include cases up to a hundred of nodes. Comparisons with other methods show a mixed behaviour: in some instances the branch-and-bound is competitive with respect to dynamic programming and other approaches, in other cases it is significantly slower, while with other instances it is able to obtain a relatively small gap between the bounds, while other approaches fail due to memory constraints.

Another structure learning algorithm of this kind is the one proposed by Malone, Yuan, Hansen, and Bridges [77], who implement a *breadth-first branch-and-bound* (BFBnB) that makes explicit use of external memory, in order to avoid the failures due to lack of RAM

for larger instances. Authors use the MDL scoring function (see section 3.1.3). In contrast with the "traditional" b&b, which performs a depth-first search if not driven otherwise, BFBnB analyzes the lattice of candidate parent sets layer by layer, in order to limit the needs for RAM; scores are stored on external memory, with a delayed removal strategy in case of duplicate scores. This slows down a little the algorithm, but ensures that the computational resources needed remain relatively low, and the execution does not fail, provided enough hard-drive space.

The bounding step is governed by an heuristic function

$$f(\mathbf{A}) = g(\mathbf{A}) + h(\mathbf{A}) \tag{3.17}$$

where $\mathbf{A} \subseteq V$ is a node of the lattice representing a subset of variables, $g(\mathbf{A})$ is the sum of costs of edges of the best path from start node to node $\mathbf{A}$, and

$$h(\mathbf{A}) = \sum_{v \in V \setminus \mathbf{A}} \text{BestMDL}(v, V \setminus \{v\}) \tag{3.18}$$

is an heuristic estimation that provides a lower bound on the cost of the path from $\mathbf{A}$ to the goal node.

3.3.5 Local learning

Niinimaki and Parviainen [82] introduce an algorithm, called SLL for *Score-based Local Learning*, that looks for the optimal Markov Blanket for each node. SLL can be divided in three steps: the first two steps reconstruct the optimal neighbourhood of each node and the optimal set of spouses for each node, respectively, by constructing and evaluating optimal networks for iteratively increasing subsets of nodes. The third step is the reconstruction of the optimal network, as adjacencies and immoralities are enough information for the task.

The authors use the algorithm of Silander and Myllymäki for finding the optimal DAGs when computing the neighbourhood and the spouses when $|V| \leq 20$, and GES for greater networks. The authors show how SLL, ran with the usual limitation of a maximum size for the neighbourhood, produces networks with lower error with respect to other approaches. On the other hand, the time complexity

is $O(|V|^4 2^{|V|})$, and is clearly dominated by the optimal network reconstruction in the first two steps. Authors report that SLL is slower than competitor algorithms.

3.3.6 Structure learning as IP problem

As mentioned previously in this chapter, learning the structure of a Bayesian Network G from data D corresponds to finding the optimal parent set $\hat{s}$ that maximizes $\text{Score}(G; D) = \sum_v \text{Score}(v, s_v)$, assuming the score is decomposable (as are commonly used scores as BDeu and BIC — see section 3.1); under this condition, the final score $\text{Score}(G; D)$ is made of the sum of local scores $\text{Score}(v, s_v)$ of the selected parent sets s_v. Being this an optimization problem, it makes sense to treat the problem as an Integer Programming (IP) problem. Two different approaches have been proposed in literature, one based on a LP relaxation (Jaakkola, Sontag, Globerson, and Meila [57], Cussens [27, 28, 26], Bartlett and Cussens [5]), and the other based on a binary vector called *characteristic imset* that looks for the Markov Equivalence classes of the DAG. Refer to section 2.4 for a brief introduction to Integer Programming.

In order to use these approaches, the log-marginal likelihood must be in linear form. For every candidate parent set, one of the possible approaches is to create variables indicating whether that parent set should be inserted in the DAG or not. Following the notation adopted by Jaakkola and Cussens, for every node v and candidate parent set s_v for v, we declare a binary variable $I(s_v \to v)$. Such variable will be set to 1 if s_v is an optimal parent set for v, otherwise it will be 0. If $\text{Score}(s, s_v)$ is the local score associated to $\{s_v, v\}$, then the objective function will be

$$\sum_{v, s_v} \text{Score}(v, s_v) I(s_v \to v). \tag{3.19}$$

All of the $I(s_v \to v)$ variables must compose a consistent set of parent sets, without any cicle. There is therefore the need to inject into the model some acyclicity constraints in order to maintain the DAG status for the graph.

The second approach, used by Hemmecke, Lindner, and Studený [54], is to find the Markov Equivalence classes of DAGs. Such approach relies on the notion of *characteristic imset*, a binary vector in-

dicating whether a subset C of nodes can be partitioned in $\{v, s_v\}$, where the nodes in s_v form the parent set of v. This latter approach relies on the property that two DAGs have the same characteristic imset if and only if they are Markov-equivalent.

Clearly, these approaches rely on an exponential number of variables, and on an exponential number of acyclicity constraints. However, it is easily observed that not all of the acyclicity constraints are really needed: for example, all of the ones that use an edge not in the Markov equivalence class, or the ones that have an edge not contained in any non-pruned candidate parent set.

In the following sections, we review the state-of-art approaches to the problem based on integer programming deeper that the other algorithms, since they form the base of the main original contribution of this thesis.

3.3.6.1 Tight LP relaxation and cutting planes

A research path is the one stemming from the PhD thesis of Sontag [102] in which a LP relaxation is adopted. Such relaxation is *tight*, meaning that the optimal solution for the primal is also the optimal solution for the dual. This framework have been described in Sontag and Jaakkola [103], Sontag, Meltzer, Globerson, Weiss, and Jaakkola [104], Jaakkola, Sontag, Globerson, and Meila [57]. A similar approach is the one followed by Cussens [27, 28, 26], Bartlett and Cussens [5], Cussens, Bartlett, Jones, and Sheehan [29].

For convenience, we will refer to the two approaches as "Sontag approach" and "Cussens approach", respectively, in what follows.

The key for the relaxation is to remove acyclicity constraints from the initial formulation of the LP problem, adding them as cutting planes when needed. Both choose the BDeu score. We use, in the following discussion, the simpler notation adopted by Cussens, as will be the same notation used in chapter 4.

Sontag approach The Sontag approach defines a family of constraints called *cluster constraints* that impose that, for every subset $C \subseteq V$ of nodes of the graph G, there must be at least one node whose parent set either completely lies outside C, or is the empty set.

Formally, the cluster constraints can be defined as follows: for every set of nodes $C \subset V$,

$$\sum_{v \in C} \sum_{s_v : s_v \cap C = \emptyset} I(s_v \to v) \geq 1. \tag{3.20}$$

This means that for every subset C of nodes, at least one of them must either be a source for the DAG, or have its parents outside C.

Clearly, there is an exponential number of such cluster constrants. The solution to this problem is to remove them, solve the linear relaxation of the problem, and look for violated constraints. Cluster constraints can be defined also for the linear relaxation, though in this form they do not suffice to shape the original polytope [57]. The approach proposed is summarized in three steps:

1. to perform a coordinate descent to solve the dual problem

$$\min \sum_{i=1}^{n} \max_{s_i \in Pa(i)} [\mathbf{c}(s_i) + \sum_{C : i \in C} \lambda_C I_C(s_i)] - \sum_{C} \lambda_C \tag{3.21}$$

$$\text{s.t.} \quad \lambda_C \geq 0, \forall C \subseteq V \tag{3.22}$$

   with a new dual non-negative variable λ_C for each cluster C;

2. decode a DAG from the current set of λ_C, and evaluate its score: if it is the same of the objective function of the dual problem, then the problem is solved;

3. if the problem is instead not solved, choose one constraint to insert into the model, and repeat the steps.

The algorithm iterates until the problem is solved, or the constraints added become too many. In this case, a branch-and-bound is performed, solving each subproblem as the main problem.

Cussens approach Cussens and Bartlett proceed in a way based on the approach outlined in section 3.3.6.1, but tackle the problem with a more traditional branch-and-cut: they consider the relaxated problem obtained by removing a more general version of the cluster constraints 3.20 from the model, and solve it, adding the most effective cluster contraints as cutting planes when needed, obtaining a solution x^*. If x^* does not violate any cluster constraint and is integer-valued, than

the problem is solved; otherwise, a variable with a non-integer value in x^* is chosen to be branched on, creating two new subproblems.

Cluster constraints are redefined in knapsack form:

$$\sum_{v \in C} \sum_{s_v : s_v \cap C = \emptyset} I(s_v \to v) \leq |C| - 1 \quad \text{where } C \subseteq V. \tag{3.23}$$

Furthermore, this family of constraints can be generalized: for every $C \subseteq V$, for every k such that $1 \leq k \leq |C|$

$$\sum_{v \in C} \sum_{s_v : |s_v \cap C| < k} I(s_v \to v) \leq |C| - k. \tag{3.24}$$

The choice of which constraints to add at every branching step is done by solving a new subproblem, looking for the constraints that may yield the better "coverage".

Since starting from a "good" point helps consistently the search for the optimal solution, the authors also propose a greedy heuristic for finding a good, albeit suboptimal, Bayesian Network, based on the observation (Silander and Myllymäki [98]) that in a DAG there must be at least one childless node, called *sink*, for which it is possible to find the optimal parents without worrying of creating a cycle.

Cussens and Bartlett provide the GOBNILP[1] package, a SCIP-based implementation of the ideas developed through their papers.

The real bottleneck lies in the precomputation of the scores for candidate parent sets, because of the exponential nature (in the number of nodes) of the search space. To overcome this issue, the precomputation is limited to candidate parent sets of bounded size (usually 3 or even 2 nodes, for larger instances). However, one can easily note that the scoring function has a sort of "bitonic" trend when varying the number of parents, therefore we can prune some of the candidate parent sets. In particular, if W, W' are candidate parent sets for v such that $W \subset W'$, $c(W) > c(W')$, then we already know that W' (and all of its supersets) will not appear in the optimal DAG as parents of node v.

As Cussens implicitly admits, requiring a bounded in-degree for the nodes prevents the optimal solution for the problem to be, in general, equal to the "real" Bayesian Network we want to learn. Moreover, there is no guarantee on the quality of the optimal solution, or

[1] http://www.cs.york.ac.uk/aig/sw/gobnilp/

even a measure of the score gap between the optimal and the real networks.

3.3.6.2 Markov Equivalence Classes via characteristic imset

Another approach comes from the PhD thesis of Lindner [73], who uses the notions of *standard imset* and *characteristic imset* introduced by Studený [109], Studený, Hemmecke, and Lindner [115] to represent the essential graph (section 2.3.3) of Markov Equivalence classes of Bayesian Networks. This approach has been subsequently studied in Studený and Vomlel [113, 114], Studený, Vomlel, and Hemmecke [116], Hemmecke, Lindner, and Studený [54], Studený and Haws [112], Studený [110, 111].

An imset is a binary vector indexed by sets of variables. Let $\mathcal{P}(V) = \{A : A \subseteq V\}$ be the powerset of V. An imset δ_A is a vector in $\{0,1\}^{|\mathcal{P}(V)|}$, $A \subseteq V$ such that $\delta_A(A) = 1$, and $\delta_A(B) = 0$ for all the $B \subseteq V$, $B \neq A$. In other words, an imset for a particular subset is a flag vector whose only non-zero component is the one indexed by the chosen subset.

Let G be a DAG, and V its set of nodes; the *standard imset* for G in $\mathbb{R}^{|\mathcal{P}(V)|}$ is

$$u_G = \delta_V - \delta_\emptyset + \sum_{v \in V} \{\delta_{s_v} - \delta_{\{v\} \cup s_v}\}. \tag{3.25}$$

Every standard imset can have at most $2 \cdot |V|$ non-zero elements, therefore it can be stored in a polynomial amount of space, despite its exponential size.

A *characteristic imset* c_G for a DAG G with node set V is a vector in $\mathbb{Z}^{|\mathcal{P}_*(V)|}$, where $\mathcal{P}_*(V) = \{A \subseteq V : |A| \geq 2\}$, computed as

$$c_G(A) = 1 - \sum_{B : A \subseteq B \subseteq V} u_G(B) \quad \forall A \subseteq V, |A| \geq 2, \tag{3.26}$$

where $u_G(B)$ is the standard imset computed in 3.25.

Imsets are related to scoring criteria via the following property ([109, 73]). Let G be a DAG over V, $\mathcal{P}(V)$ the powerset of V, $\mathsf{Score}(\cdot)$ a decomposable scoring function, u_G a standard imset for G. Then, let $\mathsf{DATA}(V, d)$ a collection of databases of length d over V, and $D \in \mathsf{DATA}(V, d)$ a database. It is possible to define a unique

mapping $t : D \mapsto t_D \in \mathbb{R}^{\mathcal{P}(V)}$ such that the resulting vector t_D contains the scores associated to the relative index of the standard imset. Then, the score of the overall network G computed over a dataset D is

$$\text{Score}(G, D) = t_D(V) - t_D(\emptyset) - \sum_{i=1}^{|u_G|} t_D(i) u_G(i). \qquad (3.27)$$

Alternatively, we can compute the score for G, D using a revised data vector computed by combining scores contained in t_D and the characteristic imset. The final transformation has the same form.

Both the standard and the characteristic imsets suffice to define a polytope, enabling the retrieval of the optimal vertex via mathematical programming tools such as the simplex method.

Characteristic imsets hold some remarkable properties related to graphical models. As said, $c_G(A) \in \{0, 1\}\, \forall A \subseteq V, |A| \geq 2$. Let G be a DAG with characteristic imset c_G, and $A \subseteq V$ a set of nodes: $c_G(A) = 1$ iff $\exists\, v \in A$ s.t. $A \setminus \{v\} \subseteq Pa_G(v)$. Two graphs G, H with characteristic imsets c_G, c_H respectively are equivalent iff $c_G = c_H$. As a consequence, if a, b, c are three distinct nodes of G, a, b are adjacent if and only if $c_G(\{a, b\}) = 1$, and $a \to c \leftarrow b$ is an immorality in G if and only if $c_G(\{a, b, c\}) = 1 \cap c_G(\{a, b\}) = 0$; also, the two last conditions imply $c_G(\{a, c\}) = 1$ and $c_G(\{b, c\}) = 1$. This is sufficient to reconstruct an essential graph from a characteristic imset (see for example Studený and Vomlel [113]). Then, we can apply a theorem from Meek [79] that shows how to reconstruct the directionality of undirected edges of the essential graph, according to the rules provided in figure 3.2.

Another result directly following by the former properties states that the characteristic imset c_G of a graph G over V can be uniquely determined by subsets of V of cardinality 2 and 3. Thus, characteristic imsets for graphical models have length polynomial in $\binom{|V|}{3}$. For forests, complexity is polynomial in $\binom{|V|}{2}$.

3.3.7 Hybrid methods and other approaches

Hybrid methods are approaches that employ both conditional independence tests and scoring functions. Independence tests are used in

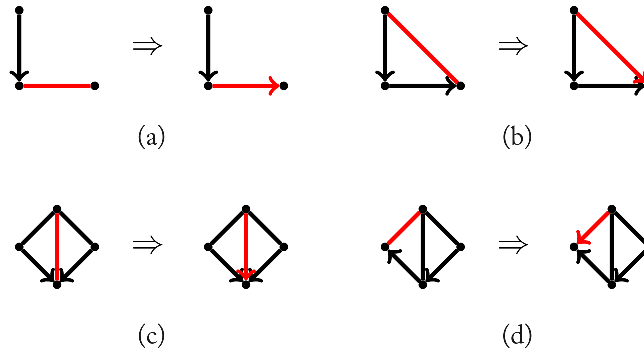


Figure 3.2: orientation rules for patterns (from [79]).

order to limit the search space, to which subsequently apply scoring methods in a more effective manner.

An example of hybrid method is the one proposed in Venco [121], Badaloni, Sambo, and Venco [4]. Their method, called CBB for *Constrained branch-and-bound*, takes advantage of the chance of injecting prior knowledge into the branch-and-bound of de Campos and Ji [34] (section 3.3.4), by means of the edge constraint matrix; such matrix is compiled by running the Max-min heuristic of Tsamardinos, Brown, and Aliferis [120] (section 3.3.3) and forbidding all of the edges that are not allowed by the resultant CPCs. In order to minimize the risk of letting out edges that are in reality present, that is, in order to recover possible false negatives issues from the Max-min step, the hill climbing step is also performed after the branch-and-bound, to explore the neighbourhood of the solution for a better DAG.

Another one is H2PC (for *Hybrid HPC*) of Gasse, Aussem, and Elghazel [46]. In similar guise of MMHC (section 3.3.3), it is based on a subroutine called HPC, for Hybrid Parents and Children, to retrieve the skeleton of the Bayesian Network; HPC mixes incremental and divide-and-conquer constraint-based approaches. Then, H2PC explores the possible DAGS over the skeleton with the same hill climbing strategy of MMHC.

Hybrid methods, since are designed to exploit the good sides of various ideas, are reported to overcome the single-method algorithms they stem from.

Alternative approaches comprise Ant Colony Optimization (De Cam-

pos, Fernandez-Luna, Gámez, and Puerta [37]), Genetic algorithms and Simulated Annealing de Campos and Huete [36].

3.4 Comments

Several different approaches have been reviewed, some other have been left out, many others are expected to be proposed, especially hybrid ones. In table 3.1 there is a summary. We can see some common points, some differences, the strength and the weaknesses for them; here we discuss the most important ones.

Exact discovery is the process of learning the network that yields the better representation for the causality relationships present in the dataset. The meaning of "better representation of the causality in the dataset" is explained in section 3.4.1. Exact discovery, however, entails heavy computational costs in order to process all the search space of the structures to analyze (usually, candidate parent sets), and are therefore limited to relatively small networks (20–30 nodes). Methods that explore a subset of the search space can achieve some improvements, but of course there is no guarantee about that. Therefore, given the size of the instance, the time and the computational resources available, one may prefer to use an approximate method. Such approximate methods, however, do not have any theoretical guarantee on the quality of the results, or the time needed to converge to a feasible solution. For this reason we may want to choose an anytime algorithm, in order to always have a feasible solution in hand.

When exploring the search space, despite some useful properties that can help us in avoiding some unfruitful areas, we necessarily have to cope with its combinatorial explosion. To cope with instances with more than 30 nodes, we have to find some strategy, even if suboptimal. Two ideas have been used for this purpose: limiting the maximum cardinality for parent sets, or retrieve an intermediate structure that allows us to maintain the conditional dependency relations among variables. Overcoming this issue is one of the keys to improve strategies of network learning.

Let k be, in this discussion, the maximum cardinality allowed for candidate parent sets. Such a limitation is often imposed by the authors due to the exorbitant computational costs of the problem: k will limit the size of the search space of candidate parent sets to be no

Table 3.1: summary of structure learning algorithms and strategies. Authors are the proposers of first implementation for each method, if further improvements have been later proposed.

Authors	Strategy
Koivisto, Sood	Dynamic programming for posterior edge probability
Ott, Imoto, Miyano	Dynamic programming for posterior edge probability
Silander, Myllymäky	DP based on variable ordering
Singh, Moore	DP based on variable ordering
Meek	Greedy edge operation selection
Tsamardinos, Brown, Aliferis	Heuristic for skeleton discovery + hill climbing heuristic for edge directionality retrieval
De Campos, Zeng, Ji	B&B with relaxation over acyclicity constraints
Malone, Yuan, Hansen, Bridges	Breadth-first B&B
Niinimaki, Parviainen	Reconstruction from optimal neighbourhood
Venco, Sambo, Badaloni	B&B over skeleton
Gasse, Aussem, Elghazel	Hill climbing over skeleton
Jaakkola, Sontag, Globerson, Meila	Linear Programming
Cussens	Linear Programming
Lindner, Hemmecke, Studený	Linear Programming
De Campos, Fernandez-Luna, Gámez, Puerta	Ant Colony Optimization
De Campos, Huete	Genetic alorithm
De Campos, Huete	Simulated Annealing

more than $\sum_{i=0}^{k} \binom{n-1}{i}$. Low values of k will therefore provide a limit over the time needed to explore this search space, an operation that many times (for many algorithms) has to be done *exhaustively*. High values for k, that is, values well beyond the maximum in-degree of a "real" network, will result in an unnecessary work done over areas of the search space that represent no realistic candidate parent sets, that are parent sets that entail dependence assumptions that are not really present in the generating distribution, with the risk of overfitting the network to the sampled data.

Bounding k is therefore an easy way to address these issues. However, k is obviously another parameter that needs to be tuned, clearly not a comfortable task, in general. A low value of k will lead to a network that will be, in general, "too simple" with respect to the original distribution of the data. A high value for k, instead, brings unnecessary work also over the tuning step.

Furthermore, a value of k imposed over candidate parent sets for all the nodes in the graph provides an "unrealistic" general limitation, in the sense that more interconnected subsets share the same constraints of sparser areas of the network; no local properties are exploited. As Tsamardinos, Brown, and Aliferis [120] point out, a node with some parents missing due to the k-bound introduces an error that propagates through the subnetwork downside the node. It has to be said that this also happens when starting the reconstruction from a wrong skeleton.

Finding a k that yields a fair tradeoff between correctness and computational time would be good. Limiting the search space in a different way, possibly locally, is even more desired.

The other widely adopted idea is to reconstruct a structure that allows us to retrieve the optimal DAG from it. Such structure may be a DAG to use as starting point for a local search; more often, it is an undirected graph, or a partially directed one. Many algorithms start their search by looking for the skeleton of the optimal DAG, and then choose some more or less clever method in order to assign a directionality to the edges of the skeleton. This strategy allows to limit the search for candidate parent sets for a node v to the set of nodes directly connected to v in the skeleton, which can be a substantially lower number of possibilities to test. On the other hand, if the pruning step leaves out one real parent, the optimal network is clearly impossible to discover. Furthermore, the skeleton discovery

step is another subroutine that needs to be tuned: a tighter pruning will yield a narrower search space, with a higher risk of losing the global optimum, while a looser evaluation allows a broader set of alternatives, entailing at the same time more time needed.

Newer algorithms tend to follow this approach, instead of blindly impose an unique limit over all of the nodes in the network. It has to be said that we do not escape the need of repeatedly scan the whole database, as this complexity is only moved to the skeleton reconstruction step. However, it seems a more reasonable one than its alternative.

3.4.1 On the significance of the results

By "significance of the results" we mean how much we can trust the reconstructed network: while the distance from a known test network can be measured, if we are discovering a new one we have no safety net. The significance is affected by several intertwined factors; we review some of them (in no particular order). First of all, the adoption of an exact algorithm or an approximate one, whose impact has already been discussed.

The second factor, as anticipated above, is the choice of the scoring criterion, if one is employed, or the conditional independence tests parameters. Conditional independence heavily rely on the tuning of their parameters, for example the threshold α we use to evaluate a conditional independence hypothesis against the null hypothesis to compute the p-value. As for the scoring criterion, we have already seen how information theoretic-based methods can help avoiding evaluating candidate parent sets, according to the Ockham's razor. However, such philosophic concept has to be considered just a "rule of thumb", or we could say an *heuristic*, in computer science terms. While introducing complexity penalty is a sensible idea, how to choose this penalty seems an arbitrary decision, though the early pruning property that derives is useful. Several studies have been published in different years, with constrasting results: Yang and Chang [123], Carvalho [16], Kasza and Solomon [61], Liu, Malone, and Yuan [75]. So, the problem of what scoring function to use is far from solved.

Finally, the third factor is the quantity and the quality of the data we are learning from. Given "wrong" data, we may have exact networks that perform worse that approximate ones. Obviously, we need

the data to be unbiased, or, no matter how good our algorithm is, or how much time we let it run, the reconstructed network will be a wrong network. Few items in the database will result in low attendibility of the reconstructed network[2], yielding also the risk of falling into a biased distribution. Missing data, and especially biased missing data, will also jeopardize the reconstruction task. Conversely, the size of the dataset is directly correlated with the resulting computational cost of the learning, so, even if we had one, we often cannot afford to use an entire huge database, and we have to subsample it while keeping the error under control.

[2]Apart from common sense, also remember the soundness property we look for in the algorithms, that holds for a quantity of data tending to be infinite.

Alternative Integer Programming formulation 4

Pros and cons of the approaches proposed in the literature have been reviewed in the previous chapters. We move from those considerations to develop an approach that can overcome at least some of those issues. We have described the scoring-based structure learning problem as an optimization problem, and have noted how few recent approaches explicitly encode the problem following one of the "natural" ways to address such kind of problems, namely linear programming. We have discussed the choice of the authors of those approaches of imposing an upper limitation over the candidate parent set size; therefore, this is our starting point for the search of a better performing algorithm. We also noticed how a recent trend in structure learning algorithms is to look for the advantages of both scoring methods and independence-based approaches, yielding to hybrid solutions; inspired by them, the idea we develop in this chapter consists in applying integer linear programming techniques over a reduced search space, in order to address the previously observed limitations of existing approaches.

4.1 Reducing the search space

First of all, the main liability of the existing approaches based on integer programming can be addressed by previously recovering an "intermediate" structure, such as a CPDAG or a Markov Network, as we have seen in algorithms such as those presented in sections 3.3.3, 3.3.7. We choose to address the issue of limiting the search space by recovering the skeleton of the network as first step, using the MMPC algorithm.

Some quick calculations can justify this choice. A network with 20 nodes and a fixed maximum cardinality for candidate parent sets of 3 yields $20 \cdot \sum_{i=0}^{3} \binom{19}{i} = 23200$ indicator variables $I(W \to v)$ for parent sets. If the maximum allowed CPC cardinality is raised to four, the number of such variables goes up to 100720, the vast majority of which is useless, as this number includes candidate parent sets formed by nodes from different areas of the network, and that would be discarded by a "semantic" analysis of the network.

Conversely, by knowing the skeleton of the network we can limit our analysis to $\sum_{i=0}^{\deg_{Skel}(v)} \binom{\deg_{Skel}(v)}{i}$ for every node v of the network, where $\deg_{Skel}(v)$ is the degree of node v in the skeleton; and furthermore, all of these variables are variables that aren't trivially prone to be discarded. The actual number depends on the instance, and may still be relevant for highly connected networks as the number of variables to examinate is $2^{\deg_{Skel}(v)}$ for each node v, but for sparser networks it may reduce to a handful of variables, not to mention that the reconstructed network will in general adhere to the real one much more than the one found by the competitor approaches.

The skeleton discovery step has however some possible downsides. First of all, this operation takes an amount of time that depends on various factors, such as the number of observed variables, the number of parameters of the variables, the connectivity of the network and the number of instances in the dataset, and it can become a bottleneck for the whole structure learning problem. All of those factors also impact the quality of the returned skeleton; for example, a small dataset will likely lead to a wrong skeleton. This step, therefore, can introduce an error that cannot be corrected in the following passages, leading to a wrong Bayesian Network. The MMPC algorithm we are going to use cannot evaluate or even recognize this error, and therefore it cannot

provide any guarantee about the correctness of its results.

In the remainder of this work, we will not address the problem of improving the MMPC step, or the development of an alternative algorithm. We will instead focus on a new modelization of the problem, based on the skeleton of the network.

4.2 From sets of nodes to edges

Our approach is to combine the reconstructed skeleton of the network with the integer linear programming approach as exploited in chapter 3.3.6. Of course, this just means to assign a consistent set of directionalities to the edges of the skeleton, in the same guise of the MMHC algorithm.

The first step is to make the edges to appear explicitly in the model, to clarify their contribution to the final score of the network. Recall the objective function implicitly used by Jaakkola, Sontag, Globerson, and Meila [57], Bartlett and Cussens [5], Hemmecke, Lindner, and Studený [54]:

$$\max \sum_{v,s_v} \mathrm{Score}(v, s_v) I(s_v \to v) \tag{4.1}$$

where the v are the nodes, the s_v the candidate parent sets, $\mathrm{Score} \cdot$ is the outcome of the score function and the $I(s_v \to v)$ are the indicator variables for *groups of edges*. We introduce a set of similarly defined indicator variables $I(u \to v)$ for edges, such that

$$I(u \to v) = \begin{cases} 1 & \text{if edge } (u \to v) \text{ is in the final DAG} \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

Note that there can be at most $|V|(|V| - 1)$ of such variables, all of the possibly defined oriented edges in a graph of $|V|$ nodes. In practice, we expect them to be much less than this number. We can make the indicator variables for edges to appear explicitly in the model by noting that:

- if the parent set s_v for node v is composed by, for example, three nodes $s_v = \{x, y, z\}$, than the indicator variable for $(s_v \to v)$ corresponds to three indicator variables $I(x \to v)$, $I(y \to v)$, $I(z \to v)$ all being set to 1; their product (logical and) will also be 1;

- at the same time, let $s'_v = \{a, x, y, z\}$ be a candidate parent set to be discarded: while indicator variables $I(x \to v)$, $I(y \to v)$, $I(z \to v)$ will be set to 1 in an optimal solution, we also want $I(a \to v) = 0$, since node a is not in the optimal parent set for node v; then, the product will be nulled by the "extra" variable;
- finally, consider the subset of the optimal parent set $s''_v = \{x, y\}$: being composed by two nodes whose corresponding indicator variables (towards v) will be set to 1, then also the term $\text{Score}(v, s''_v) I(s''_v \to v)$ will be included in the cost of the objective function; and while this does not compromises the optimality of the solution computed, since the same set of directed edges will be selected, its cost will be higher than the true optimal one.

The first two observations allow us to replace

$$I(s_v \to v) = \prod_{w \in s_v} I(w \to v) \tag{4.3}$$

for all the nodes v, candidate parent sets s_v for v. Such product will be 1 only if the all the nodes in s_v really belong to the optimal parent set for v. The third observation, instead, suggests to discard subsets of the optimal parent set. However, if the previous substitution can be trivially done without any risk, this one needs a little more attention: we do not know in advance which is the optimal parent set, so we do not know whether a set of nodes is an optimal parent set, a superset of the optimum, or a subset of it. Recall that we are orienting the undirected edges of the skeleton; so, the set of nodes connected to node v form its CPC (Candidate Parent-and-Children, see section 3.3.3). We want only one subset of nodes $s_v \subseteq CPC(v)$ to be selected, and the other ones to be discarded; that is, only one product to be 1, and the other ones to be brought to 0 by the variables for the edges linking v to his children. This can be achieved by adding a "reversed" indicator variable for edges from v to his children, and can be simply done by taking $1 - I(w \to v)$ into the product for $w \notin s_v$, for all the subsets $s_v \subseteq CPC(v)$. So, the previous substitution 4.3 can be replaced by

$$I(s_v \to v) = \prod_{w \in s_v} I(w \to v) \prod_{w \in CPC(v) \setminus s_v} (1 - I(w \to v)). \tag{4.4}$$

The second term zeroes all of the contribution other than the one for the optimal choice: let $CPC(v) = \{w_1, \ldots, w_k\}$ be the candidate

parent-and-children from the MMPC algorithm, and s_v a subset of it; then

- if $s_v = \{w_1, \dots, w_j\}$, $1 \leq j \leq k$, is the optimal parent set, then the indicator variables for the edges coming from the optimal parent set (those we want to be 1) will be in positive form, while the variables we want to be zero will be in the second product in reversed form, thus also contributing as 1s in the overall product. This also works if the optimal parent set is the empty set (node v is a source for the network);
- if s'_v is a subset of the optimal parent set, there will be at least one exceeding variable in the second product, namely the variables corresponding to the edges from the nodes in $s_v \setminus s'_v$ to v; such variables will therefore be brought to zero, and will nullify the overall product;
- if s''_v is instead a superset of the optimal parent set s_v, there will be at least one exceeding variable in the first product, namely the variables all the variables corresponding to the edges from the nodes in $s''_v \setminus s_v$ to v; those variables will contribute with a zero in directed form;
- the last case is the case where a subset s'''_v of $CPC(v)$ both contains some children of v and misses some parents of v; the effect is the combination of the second and the third cases.

Therefore, by all of the previous considerations, we could rewrite the objective function as

$$\max \sum_{v, s_v} \text{Score}(v, s_v) \left(\prod_{w \in s_v} I(w \to v) \prod_{w \in CPC(v) \setminus s_v} (1 - I(w \to v)) \right). \tag{4.5}$$

Unfortunately, such objective function is nonlinear. Therefore, we have to manipulate it in order to make it linear. First of all, we step back to the original objective function 4.1, and insert equations 4.4 in the model as constraints, one for each subset of variables. Such *bounding constraints* are nonlinear but can be easily converted in a set of linear constraints using a generalized techniques for linearizing a product of binary variables using additional constraints and an additional variable for each product. In our case, however, there is no

need for such additional variables, since their role is played by the indicator variables for subsets. The product $a \cdot b$ of two binary variables $a, b \in [0, 1]$ can be defined with a new variable z, and the following set of constraints:

$$z \leq a \quad \Rightarrow \quad z - a \leq 0 \tag{4.6}$$
$$z \leq b \quad \Rightarrow \quad z - b \leq 0 \tag{4.7}$$
$$z \geq a + b - 1 \quad \Rightarrow \quad z - a - b \geq -1. \tag{4.8}$$

Constraints 4.6 and 4.7 ensure z to be 0 if at least one of the product factors is 0, and therefore we shall call them *lower-bound constraints* for the product variables in this context; inequality 4.8, instead, induces $z = 1$ if all of the factors are 1, and we call it an *upper-bound constraint*. Furthermore, this set of constraints will be satisfied only all of a, b, z have the same value (0 or 1); the truth table for a, b, z is the proof of correctness. Because of the form these constraints will assume when inserted in a LP solver, we will often prefer to write all the variables in the left-hand-side of the inequalities, and the constant terms in the right-hand-side.

The case of a product with three binary variables a, b, c is treated equivalently:

$$\begin{aligned} z - a &\leq 0 \\ z - b &\leq 0 \\ z - c &\leq 0 \\ z - a - b - c &\geq -2. \end{aligned}$$

Using our variables, if $s_v = \{x, y, z\}$ is a candidate parent set for v, we have

$$\begin{aligned} &I(s_v \to v) - I(x \to v) \leq 0 \\ &I(s_v \to v) - I(y \to v) \leq 0 \\ &I(s_v \to v) - I(z \to v) \leq 0 \\ &I(s_v \to v) - I(x \to v) - I(y \to v) - I(z \to v) \geq -2. \end{aligned}$$

Every set of edges s_v induces no new variables in our case, since we already needed to define indicator variables $I(s_v \to v)$, and $|s_v|$ new constraints. In our case, we have both variables in directed and

complemented form, but this adds very little complexity to the problem: for example, we could simply create one new variable for each variable in complemented form. This solution would add as many new variables and constraints as the number of variables in complemented form. Otherwise, we can just iteratively build up the new bag of constraints by adding, for each factor, a new lower-bound constraint, a variable in the upper-bound constraint, and its contribution to the constant term in the right-hand-side of the upper-bound constraint.

Furthermore, the number of inequalities to be added to the model for candidate parent sets of cardinality greater ≥ 3 can be further reduced down to two, by noting that the lower-bound constraints can be summed into one of the form

$$k \cdot z - a - b - \cdots \leq 0, \tag{4.9}$$

where k is the number of factors in the product. Again, a simple truth table, being $a, b, \ldots, z$ binary variables, ensures the correctness of this derivation.

It remains one little issue to solve: as the scores of candidate parent sets are computed as the logarithm of a probability, their value is negative. The overall value of the objective function is therefore maximized if no candidate parent set is chosen, clearly a solution we cannot accept. We therefore need to reintroduce the convexity constraints

$$\sum_{s_v \in CPC(v)} I(s_v \to v) = 1 \quad \forall\, v \in V, \tag{4.10}$$

in order to impose that one (and only one) candidate parent set for each node is selected in the optimal solution (remember that the empty parent set is also present in the CPC).

This way, we have explicitly related each set of candidate parent sets to its edges, at the cost of an increment of the number of constraints of $2 \times |CPCs(v)|$ for every node v. In theory, there may be an exponential number of these bounding constraints. In practice, their number depends on the sensitivity of the parameters in the MMPC step, or any other algorithm we use to build up the skeleton, and we expect them to be not too many.

We have also to impose that no undirected edge of the skeleton can be inserted in the DAG with both directions enabled. Equivalently, every undirected edge in the skeleton must be assigned a direc-

tionality edge. This is translated with a set of *directionality constraints* of the type

$$I(u \to v) + I(v \to u) = 1 \quad \forall\ (u, v) \text{ in the skeleton.} \qquad (4.11)$$

Clearly, there are at most $|V|(|V|-1)/2$ of such constraints, but very likely in an efficient implementation they would be much less than the theoretical maximum number, since, as before, the purpose of the skeleton reconstruction is to effectively prune the edge search space.

However, the problem is not wholly defined. So far, the model composed by objective function 4.1 and constraints 4.6–4.8, 4.11 does not prevent directed cycles in any feasible solution. We can reintroduce the cluster constraints 3.20 from section 3.3.6.1, or their more general form 3.23, and have a complete model for the directionality reconstruction problem. Another possible idea is to avoid cluster constraints and exploit two elementary facts: the skeleton already contains all of the possible cycles we can end up with, and such cycles must have one very simple property, when assigned a directionality to their edges, in order to form a DAG.

4.3 A family of skeleton-based cuts

Under this model, we can adopt an alternative family of cuts with respect to the previous works, whose purpose is to help the process of assigning a consistent directionality to the edges in the skeleton. The purpose of these cuts is to eliminate solutions that contain directed cycles; however, they work also for undirected cycles, so that we can add them when evaluating the skeleton. We define them for the directed case; as for undirected cycles, it only suffices to treat them as two different directed cycles defined over the same sequende of nodes.

Recall that we have introduced indicator variables $I(u \to v)$ for the edges: then, a directed cycle of length k will be denoted as a sequence of k edge indicator variables $I(v_1 \to v_2), I(v_1 \to v_2), \ldots, I(v_k \to v_1)$. In order to break it, we can adopt common acyclicity constraints in the form of knapsack constraints

$$I(v_1 \to v_2) + I(v_1 \to v_2) + \cdots + I(v_k \to v_1) \leq k - 1. \qquad (4.12)$$

These constraints are not new, as are popular in other graph problems such as the TSP (see for example Applegate et al. [3]). Their advantage with respect to the cluster constraints of Jaakkola et al. [57] lies in their immediate detection and insertion, requiring to consider only the edge indicator variables, and not on the parent set indicator variables. The separation problem is therefore much easier, being restricted to a depth search in a graph.

4.3.1 Finding violated cuts

By solving the model, we obtain the parent sets for the nodes, and the adjacency matrix of the graph, that we suffice to check in order to check if the returned graph is indeed a DAG. Exploiting the fact that a DAG, and every of its sub-DAGs, must have at least a sink node, we iteratively shrink the matrix, removing the rows and columns of the sink nodes. If the graph is acyclic, we will eventually eliminate all of the matrix; conversely, the nodes forming a directed cycle will prevent us to empty the matrix, and we need to identidy these nodes and the edges between them.

The literature about finding cycles in directed graphs is not as rich as it is for undirected graphs; we adopt a simple heuristic that finds at least one cycle, even if we are not guaranteed to detect all of them. Starting from a node v, we try to return to it with a depth-first search. If we cannot reach again v from itself, then the node is not part of any cycle. Otherwise, we track the sequence of edges traversed, to impose a knapsack constraint 4.12. We repeat this quest starting from every node of the graph. Since this naive approach will find a cycle many times (precisely, a number of times equal to the number of nodes it is composed of), as we have found a cycle, we temporarily remove one of its edges from the graph; this prevents to find the same cycle again, but also may prevent to detect other different cycles that traverse the same edge, hence the heuristic behaviour.

4.4 Notes on the model

We have provided a model that fully defines the structure learning problem as an integer linear programming problem. We have formulated it in terms of looking for the best edge orientatation over a

skeleton, but it is almost valid for an edge orientation step defined over any undirected structure, even a complete graph. We only need to enable the possibility of discarding an edge if the solution does not require it: therefore, we redefine directionality constraints 4.11 in

$$I(u \to v) + I(v \to u) \leq 1 \quad \forall\ (u, v). \tag{4.13}$$

This also serves as a correction to the skeleton discovery step, allowing the solver to discard edges incorrectly selected by the MMPC step, or by any other algorithm we choose for the first phase. It is therefore advisable to use this version of directionality constraints, instead of the former one.

The idea behind this relaxation is to allow more freedom to the LP solver, hoping this can provide quick improvements for the objective function by selecting only the edges that really improve the global score. This solution has also the advantage of rewarding simpler networks, partially addressing that issue that motivates the penalty term in information-theoretic scoring methods.

We have defined our model essentially by extending the model used by Jaakkola, Sontag, Globerson, and Meila [57], Bartlett and Cussens [5], embedding the adjacency matrix in a framework that considers only candidate parent sets. Therefore, all of the theory developed by them is valid also in our model. At the same time, from our model we have formulated a solution process that aims at discovering the same network the MMHC algorithm looks for: the hill-climbing orientation is just one of the possible approaches to solve a linear model. We have therefore formulated a general model for the structure learning optimization problem, that bridges several methods previously proposed in literature.

The linear programming formulation also provides a clear representation of prior knowledge. Any notion we already have can be injected into the model simply by introducing an appropriate constraint: for example, for mandatory or forbidden edges we suffice to set the relative variable to 1 or 0, for sources or sinks of the network we just need to impose or deny CPC variables, an ordering of the nodes is representable by denying the edges and CPCs that violate it, and so on.

4.4.1 Relations with other problems

Other authors (Koivisto and Sood [66], Larranaga, Kuijpers, Murga, and Yurramendi [70]) have previously noted how the structure learning task resembles the Traveling Salesman Problem [3]; Sahai, Klus, and Dellnitz [94] have suggested a procedure to order the nodes in a Bayesian Network by transforming it into a TSP instance. As the TSP is surely one of the most studied problems in computer science, combinatorics and optimization, it would be interesting to find some corroboration to this intuition. Trivially, we can say that, being them two NP-complete problems, we can surely transform one into the other. We add some further considerations, in order to understand how to accomplish this.

First of all, the acyclicity constraints 4.12 are exactly the same acyclicity constraints that can be defined for the Asymmetric TSP; however, the subtour elimination constraints for the Symmetric TSP have the same form. Furthermore, the cluster constraints 3.20 by Jaakkola et al. [57] resemble the TSP acyclicity constraints in flow form, where the edges are thought as a network, and some flow has to go from some node to all of the other nodes, exiting any cluster of nodes smaller than the whole graph.

We argue that the model for the BN structure learning problem can be translated into a model for the Generalized Traveling Salesman Problem (Srivastava, Kumar, Garg, and Sen [108], Laporte and Nobert [69]), where the nodes are partitioned into clusters and each cluster has to be traversed one and only time[1]. A GTSP instance can be transformed into a TSP instance (Noon and Bean [83], Fischetti, González, and Toth [43]).

One of the possible ILP formulations for the GTSP over an undirected graph $G = (V, E)$ with n nodes partitioned into m clusters is the following [43]:

$$\min \sum_{e \in E} c_e x_e \tag{4.14}$$

[1]The GTSP also has a more general formulation, where each cluster can be traversed multiple times, with the constraint 4.16 being ≥ 1. The version of GTSP we employ is sometimes referred to as EGTSP.

subject to

$$\sum_{e \in \delta(v)} xe = 2y_v \qquad \forall v \in V \quad (4.15)$$

$$\sum_{v \in C_h} y_v = 1 \qquad \text{for } h = 1, \dots, m \quad (4.16)$$

$$\sum_{e \in \delta(S)} xe \geq 2(y_i + y_j - 1) \qquad \forall S \subset V, 2 \leq |S| \leq n-2,$$

$$\text{for some } i \in S, j \in V \setminus S \quad (4.17)$$

$$x_e, y_i \in \{0, 1\} \qquad \forall e \in E, v \in V \quad (4.18)$$

where c_e is the cost of the edges $e \in E$, the x_e are binary indicator variables for the edges, being them set to 1 if the edge is selected in the optimal solution and 0 otherwise. The variables y_v are indicator variables for the edges (in flow fashion) respectively leaving and entering each node; constraints 4.15 and 4.16 ensure that for each cluster S strictly smaller than V there is at least one exiting edge. The inequality family 4.17 are the subtour elimination constraints. Finally, all the indicator variables are binary.

We move from this observation: the nodes in the GTSP instance are partitioned into clusters, and only one node for every cluster can be selected, just like only one candidate parent set can be selected for every node of the DAG. The GTSP solution must obey to some constraints, whose form is not too different from the form of the cluster constraints proposed by Jaakkola, also in their equivalent knapsack formulation as proposed by Cussens; furthermore, also the constraints 4.12 defined over the set of edge indicator variables are a family of basic subtour elimination constraints for a TSP problem. This suggests to relate the candidate parent sets for the nodes in the DAG to the nodes in the GTSP instance: the optimal tour for the latter will be composed by the nodes corresponding to the candidate parent sets selected to form the optimal DAG. The convexity constraints 4.10 correspond to 4.16 and ensure that only one node in each cluster is selected in the tour.

The main discrepancy is the fact that the scores in a BNSL instance are defined over the nodes of the DAG, while a GTSP instance evaluates the cost of the edges to be selected. In the GTSP, however, the choice of the edges is related to the choice of the nodes to be connected: different sets of edges imply different sets of nodes. The

different objective function direction, once the different evaluation target is settled, is not an issue, as the objective function of GTSP is (related to) the dual of the one for BNSL. A proposal for the transformation is the following: since we have to maximize the sum of negative scores defined over sets of nodes, we assign to each edge of the GTSP instance a cost computed as the sum of the scores of the nodes they insist to. Then, by considering the opposite of this cost we have transformed the problem into a minimization problem over the cost of edges.

We investigate now the connection among the acyclicity constraints in BNSL and GTSP. Let's consider a complete model for the structure learning of a DAG over a graph $G = (V, E)$ with $n = |V|$ nodes, without any preprocessing, therefore including all of the $n \times 2^{n-1}$ candidate parent set indicator variables (as mentioned earlier, the n^2 edge indicator variables can be omitted from this discussion), each with its score, and all of the acyclicity constraints in it. In order to translate one instance into another, we have to show that the constraints 3.20 suffice to define the acyclicity constraints also in a GTSP instance $H = (W, F)$ with $m = |W|$ nodes partitioned into n clusters of 2^{n-1} nodes. Constraints 3.20 state the following: for every a subset $C \subset V$, there must be at least one node $v \in C$ whose parent set either lies in $V \setminus C$ or is the empty set (v is a source for the DAG). In other words, among the candidate parent sets s_{v_C} for nodes $v \in C$, at least one must be formed of nodes not included in C. If we assign a candidate parent set s_v for the nodes in V to every node $w \in W$, it means that in a set of nodes $D \subset W$ selected according to some rules, at least one node $w' \in D$ must be selected in the optimal tour. Those rules are just the acyclicity constraints.

So far, the conversion is straightforward. However, with this transformation, the resulting GTSP instance may have an optimal solution that corresponds to a non-feasible instance of the BNSL instance. We may consider, at this point, an hybrid model to be solved with a cutting plane strategy, initially converted into GTSP and solved until the optimum; then the solution is checked against the acyclicity constraints for BNSL, the violated ones are added into the model, and such a procedure iterates until a valid solution for BNSL has been found.

We have proposed a pathway to follow to link the problem we are studying in this thesis to a more studied one. However, we have

stopped halfway, and have failed in providing a full, polynomial-time reduction among them, having found no proof of correctness for our transformation, but only some intuitions. We still hope to have provided some support to the intuitions of Koivisto and Sood, and Larranaga *et al.*, and to the TSP-based approach of Sahai, Klus and Dellnitz.

4.5 A cutting-plane algorithm

The number of variables and constraints and their computational requests are factors that have an impact over the performances of a solver. Of course, the more constraints we add, the more computation is requested in order to validate the solutions, the more time it will take to discover and certify an optimal solution. This consideration is the one that leads to the cutting-plane algorithm (see section 2.4.2). Since, in practice, only a fraction among all of the constraints are sufficient to shape a polytope, we obviously want to discover which are these constraints, and insert only them.

We can start from a relaxated model, without the acycliclicy constraints so that, in general, a solution to the relaxated problem may contain directed cycles. Then, with a cutting plane algorithm, we evaluate the solution, detect the directed cycles, and inject into the model the cuts that deny such cycles. Pseudocode is given in algorithm 4.1. The question becomes: are there any constraints whose effect include other constraints, that we can therefore avoid to include?

A possible solution to this problem is to insert the largest cycles first; such cycles are "outer" cycles, possibly chordal, but, likely, none of its edges will be a chord for other cycles. Its chords may then be oriented with some local method (e.g. the Meek rules)[2]. Moreover, since in general an edge may be part of more than one cycle, orienting an edge has a cascade effect over multiple cycles, allowing us to propagate the directionality over larger sub-DAGs; this may done with some custom subroutine, or by let up to the solver. Observe the example in figure 4.1: we see that for a cycle connecting a set of nodes in the skeleton, there may be some chords connecting non-adjacent

[2]Though a LP solver will solve the instance until the optimal solution is found, unless instructed otherwise.

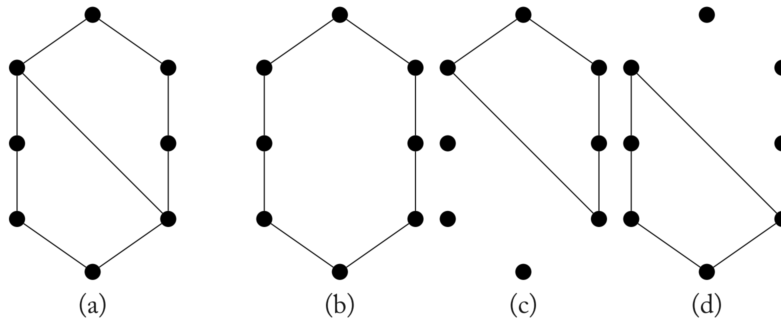


FIGURE 4.1: a set of edges forming cycles for the given set of nodes. In (a) it is the subgraph obtained in the skeleton, that contains the three cycles (b), (c), (d).

nodes in the cycle. In the example, we would have 3 acyclicity constraints inserted in the model. However, we may insert only the constraint for cycle (b), and the orientation of the chords will follow (in a custom algorithm, we may adopt for example the Meek rules).

4.5.1 Computational costs and considerations

As the problem of structure learning is NP-Hard, we have to carefully consider the computational issues in order to understand how the new formulation behaves according to the size of the instances. We consider only the directionality reconstruction, since the MMPC step is left untouched. In the following analysis, we compute the theoretical worst-case complexity for each element of the model. In practice, we expect the MMPC step to heavily trim the search space, in order to have an overall complexity parameterized in $\deg(G)$ instead of $|V|$.

The objective function 4.1 is defined for every possible subset of nodes, therefore there are $O(2^{|V|-1})$ candidate parent sets for each node. Note that this complexity is not far from the $O(|V|2^{|V|})$ of dynamic programming algorithms. We have already discussed how to adopt a lower number of variables, but the upper bound is, of course, exponential.

Also the constraints 4.4 are defined for every candidate parent set, so their number is exponential. For each of them, we need to add to the model two upper- and lower-bound constraints 4.9–4.8.

Algorithm 4.1: Cutting plane algorithm for learning a DAG from a database.

```
   Data: Database D
   Result: Optimal DAG for D
1  Skel = MMPC(D);
2  c = set of scores for candidate parent sets over Skel;
3  P = LP model over Skel, c;
4  P_r = relaxation of P, w/o source/sink constraints;
5  x̂ = solution of P_r;
6  if P_r is unbounded or infeasible then
7  |  stop;
8  end
9  C = set of violated source/sink constraint in x̂;
10 while C ≠ ∅ do
11 |  c* = most effective cut ∈ C ;
12 |  add c* to P_r;
13 |  x̄ = solution of P_r;
14 |  if P_r is infeasible then
15 |  |  stop;
16 |  else
17 |  |  x̂ = x̄;
18 |  |  C = set of acyclicity constraint violated in x̂;
19 |  end
20 end
21 return x* = x̂;
```

There is one directionality constraint 4.13 per undirected edge, therefore we need to insert $O(|V|^2)$ of them.

There is also one acyclicity constraint 4.12 for every directed cycle in the graph, therefore we need two constraints for every undirected cycle. This is the major pitfall: to its consequences and possible (partial) solutions the section 4.5.2 is dedicated.

4.5.2 Finding cycles

The number of possible cycles in an undirected graph with n nodes is (Johnson [58]):

$$\sum_{i=1}^{n} \binom{n}{n-i+1} (n-i)!,$$

clearly an intractable quantity for even small instances. However, while we need to define acyclicity constraints for every cycle in the skeleton in order to give a complete formulation of the problem, we have already discussed how we can remove them and add only some of them as cutting planes; we have also argued that if we add cuts for some cycles, its chords and adjacent cycles may get oriented by a cascading effect as well.

Anyway, the computational bottleneck has just moved to the problem of finding "large" cycles. Unless P=NP, there is no way of doing such task exactly and efficiently, as it is a modified version of the HAMILTON problem of finding a Hamiltonian circuit in a graph, both directed and undirected (Karp [60]). Thus, we have to settle for "sufficiently good" cycles, and trust in some cascading effect. In fact, as often occurs in practice, we prefer to quickly find some decent cycles, instead of spending too much time for computing the best cycles, since the effect on the subsequent iterations of the cutting plane algorithm may be negligible.

One idea is to find the chordless cycles of the graph, and add them as cutting planes. Since all the directionalities of edges in adjacent cycles must eventually agree, this approach is guaranteed to cover all the possible cycles in the graph.

Another possible idea is to look for longer cycles. We propose a simple heuristic to collect cycles up to a certain length l in an undirected graph, to use as subroutine in the cutting plane algorithm. This heuristic is nothing more than a simple modification of a breadth first search. To detect if a graph has cycles, one can start from any node, move to adjacent nodes using a depth search, and stop when a previously visited vertex is encountered. If this stop happens after say m steps, the cycle detected would be of length at most $2 \cdot m$, or $2 \cdot m - 1$ if the doubly-visited node has been joined for the first time in the current round or in one of the previous iterations, respectively. As the depth-first search may not return any cycle if stopped after some predetermined number of steps, and given that we are inter-

ested in quickly collect some cycles, we prefer to use a breadth-first search. Therefore, if we want to collect cycles of length at most l, we can allow the search to run for $\lceil l/2 \rceil$ iterations. To ensure the algorithm really looks for broad cycles, and not a small chordless one, we have to overwrite the previous node information after the cycle has been detected. Note that this may trigger some degenerate behaviour, for example when a node being evaluated is part of a clique with more than three nodes (see for example figure 4.3). The last cycle to be found can be returned to algorithm 4.1 as a candidate for a new cut. We can also choose to return the k longest cycles discovered in this way by keeping a queue of length k, in which storing the latest longest cycles found (we are not interested in short subcycles, and our heuristic will not even recognize them). Such procedure is also trivially parallelizable, by starting from different vertices; having the skeleton, we could even carefully choose the starting nodes. Note that while initially such heuristic will be called over the skeleton, in successive iterations we would have a partially directed graph in hand; no modifications are however needed in our discussion, since it would just limit the search space.

Pseudocode for this heuristic is given in algorithm 4.2, under the name of (k,l)-*BFS*, short for "breadth-first search that returns up to k longest cycles of maximum length l". Of course the "longest" attribute is just a local property among the discovered cycles, not the longest cycles in the whole graph. Some details have been left in general form, such as the queue management (e.g. removal of $(k+1)$-th oldest cycle) and the cycle identification (following backward the *prev* information while avoiding infinite loops over cliques), as are considered given subroutines.

Some improvements could be added, for example ensuring that no cycle is returned more than once, but since this has no effect over the correctness of the final result, we do not consider them.

In our approach, the (k,l)-BFS algorithm can be used only on the skeleton, since every edge will be oriented after the first iteration of the solver. When the graph becomes directed, we have to find another way to solve the separation problem. Note that we cannot use Tarjan's algorithm for strongly connected components in a directed graph (Tarjan [117]), as the final graph in general has no such components. We have already mentioned how we can try to find a path that, from every node, returns to the same node. That search will be

Algorithm 4.2: (k, l)-BFS

```
   Data: G = (V, E), node v, parameters k, l
   Result: set of maximum k cycles in G of length up to l
 1 frontier = [v];
 2 max_iters = ⌈l/2⌉;
 3 curr_iter = 1;
 4 labels = [1, ..., |V|];
 5 prev = [0, ..., 0];
 6 Q = queue of length k;
 7 while curr_iter < max_iters and frontier is not empty do
 8   | new_frontier = [];
 9   | while frontier is not empty do
10   |   | next = extract first element from frontier;
11   |   | for i ∈ neighbourhood(next) do
12   |   |   | if labels[i] == labels[next] then
13   |   |   |   | // already visited node: insert cycle in Q;
14   |   |   |   | c = identify cycle;
15   |   |   |   | insert c into Q;
16   |   |   | else
17   |   |   |   | // no cycle: continue search on this path;
18   |   |   |   | labels[i] = labels[next];
19   |   |   |   | prev[i] = next;
20   |   |   |   | insert i in new_frontier;
21   |   |   | end
22   |   | end
23   | end
24   | frontier = new_frontier;
25   | curr_iter = curr_iter + 1;
26 end
27 return Q;
```

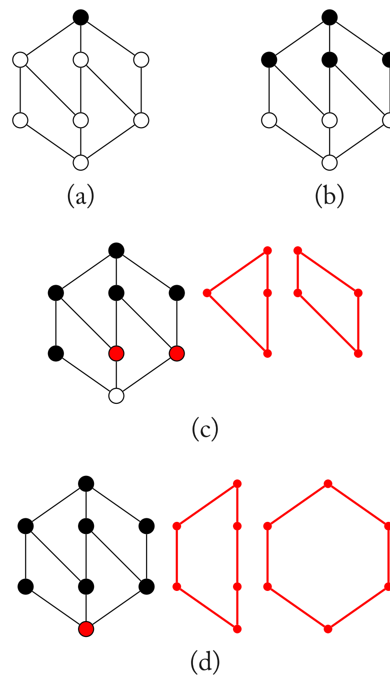


Figure 4.2: a four-iterations run of the (k, l)-BFS heuristic: black nodes are the vertices already visited, white nodes are the vertices yet to be visited, and red nodes are the vertices visited from two or paths, indicating that a cycle has been found. In (c) and (d) we see the cycles found (in red). Note that in the original graph there are cycles that are not found by the heuristic.

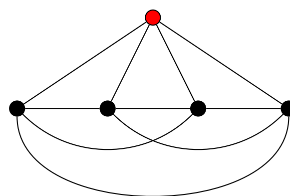


Figure 4.3: degenerate case for (k, l)-BFS to take into account for: starting from the red node, at every iteration each node overwrites the *prev* information, ending in an infinite loop.

our separation procedure.

4.6 Solving the model efficiently

A first relaxation by removing acyclicity constraints has already been exploited, and we have also provided a cutting plane algorithm to solve the problem this way. However, the performance of the algorithm are still highly dependent on a series of factors, among which the most relevant one is the degree of the nodes in the skeleton, that is the primary responsible for the number of variables, and therefore constraints, in the framework. In order to solve the model efficiently, we need to develop further considerations.

First of all, the pruning strategy devised by De Campos and Ji ([34]) finds a large number of candidate parent set that can be excluded from the search space without further consideration. This is also very appropriate to our approach of already pruning the search space by first finding the skeleton.

4.6.1 Partial linear relaxation

A node which is connected to k other nodes in the skeleton yields up to 2^k candidate parent sets. In mid-to-large network, a highly connected node, acting as "hub" for the skeleton, is likely to burden the model with thousands of parent set indicator variables $I(s_v \to v)$ and thousands of constraints relating them to edge indicator variables $I(u \to v)$; the number of edge indicator variables remains instead much lower, as there are two variables $I(u \to v)$ for every undirected edge in the skeleton. A model with thousands of integer variables is likely to take a lot of time and computational resources to be solved; we can instead solve its linear relaxation and observe the result. We restrict the continuous relaxation to the candidate parent set indicator variables, that should be the vast majority; the edge indicator variables are a limited number, and likely many of them are already ruled out by the MMPC step.

Recalling figure 2.3 we analyze what happens with this relaxation. The outcome will be a valid DAG, because the acyclicity constraints are imposed over the edge indicator variables, that are integer. However, the final DAG will also very likely be suboptimal, because its

computed cost, is (at least partially) composed of fractional contributions. This is due to the fact that, having linearized the constraints 4.4, we have lost the zero-product property entailed by the edge variables set to zero. The objective function is therefore driven towards a vertex of the relaxated polytope, which will not, in general, correspond to a point of the integral polytope. The score computed for the linear solution is an upper bound for the real optimal score of the integral solution.

We need therefore to round the fractional variables of the LP solution in order to provide a final integral solution. This is a computationally hard step, since we are enforcing the integrality property, which takes responsibility for placing the 0-1 integer linear programming problem in the NP class.The problem is, of course, that each variable entails a strong dependence over the other variables.

The LP solution will, in general, divide the parent set indicator variables in three categories: a small number of variables already set to one, a larger number of variables already set to zero, and the remaining variables being fractional. The edge indicator variables, along with the wink/source variables defined when ruling out cycles, will instead already be integer, as we have not relaxated them. We can select at least three different methods for rounding the fractional variables; we outline them in the following sections, and explain their characteristics. Clearly, the computational time needed varies according to the quality of the final solution we are going to return..

4.6.1.1 Getting CPC variables from the DAG

The first strategy we can choose is to ignore the parent set indicator variables and note that the edge indicator variables already compose a valid DAG. We can therefore reconstruct the parent sets, and compute the total score. This recomputed score will be a lower bound of the optimal integral solution, as it is a valid DAG, otherwise it would have been ruled out by some cutting plane, but we cannot, at this point, state its optimality (in fact, in general it won't be the case). Of course, we have ignored the CPC variables of the LP solution, but their connection with this DAG-based solution is easily seen: this solution will contain the candidate parent set variables that were already chosen in the LP solution, otherwise a different, better, fractional or

integer configuration would have been selected, with a different set of edges imposed in the DAG.

4.6.1.2 Iterated exhaustive rounding

Then, another idea we can try is to impose the integrality condition over the fractional variables of the LP solution, and solve again the model. What will happen is that the score will be lower than the previous one, as we have enforced a more constrained configuration, but the solution, in general, will also contain other fractional variables. The set of edge indicator variables may be the same than the set at the previous step, or may vary. We then compose the CPC variable set from the DAG, as before. This procedure can be iterated until no more fractional variables are present in the linear solution, yielding a complete evaluation of the search space.

Preliminary experimental evaluations show that while the cost computed with the linear relaxation always decreases towards the real optimal cost, the recomputed network at some generic iteration may have a cost that is worse than a previously found DAG. This yields worse results along with unnecessary computation, and we want to avoid it. The solution we can adopt is to simulate a tabu list (see Glover, Laguna, et al. [50]) by adding a constraint that denies that worse solution. We have two possibilities: to forbid the combination of the CPC indicator variables, or to forbid the combination of edge indicator variables. We prefer the second hypothesis, as a more powerful cut of integer solutions, while the first possibility is just a fractional cut, much less effective in practice. Formally, the cut we add is

$$\sum_{(u \to v) \in E^{DAG}} I(u \to v) < |E^{DAG}|, \tag{4.19}$$

meaning that the sum of edge indicator variables for the selected edges in the DAG (E^{DAG}) must be lower than the number of edges in the DAG; that is, the successively computed DAG will contain at least a different edge. If we restrict the candidate parent sets variables involved to be integer, this constraint also implies the constraint over the parent set indicator variables, because, as at least one edge is subject to change, then at least one candidate parent set is going to be ruled out. Every improving solution gets stored as incumbent. We can keep the size of the tabu list, the tabu tenure, as infinite, meaning

that we never remove this last constraint from the model. This also maintains an anytime condition for the algorithm, that is, at each moment the algorithm can return the best network discovered up to that moment.

At each iteration the problem will also be more difficult than the problems in the previous iterations, thanks to the higher number of variables required to be integer, and the higher number of constraints that will be added if the LP solver finds other cycles. Moreover, we are not guaranteed that the changes immediately improve the quality of the network, that is, its similarity to the network that would be found by an exhaustive algorithm, or by a sound algorithm with an infinite amount of data.

What we see from preliminary tests is that imposing the integrality condition over the fractional values in many cases leads the LP solver to discard those variables setting them to zero, and choose different variables, in general in fractional form, that were previously unselected. Thus, this iterated integrality enforcement pushed the objective function around the search space.

A high-level pseudocode for the oulined method is given in Algorithm 4.3. By stopping the algorithm at a different condition, for example after a given amount of time, or when a given gap between the linear solution upper bound and the integral solution lower bound is reached, we can return the best network we have at hand, that is, a suboptimal DAG.

4.6.2 Computational techniques

Apart from the ILP-based relaxations proposed so far, we note that the cutting plane algorithm requires to solve from scratch a model once it has been strengthened with the addition of cutting planes. This is computationally expensive, first because we require the solver to evaluate all the polytope again, even to move to an adjacent vertex, and then because the model we ask to solve is increasingly heavy iteration after iteration, thanks to the cuts added. In particular, the by iterating this procedure until the optima solution is found, all we do is to perform a huge, unnecessary amount of work only to solve, at the last step, just the original, unrelaxated model. This is rational only if we plan to perform few iterations of the cutting plane algorithm,

Algorithm 4.3: ITERATED INTEGRALITY ENFORCING ROUNDING ALGORITHM.

```
   Data: Problem model
   Result: Integral solution
 1 initialize problem;
 2 solve the linear relaxation of the model;
 3 let x̃ the solution obtained;
 4 best_solution = x̃;
 5 STOP = false;
 6 repeat
 7 |  x̂ = get_CPCs_from_DAG(x̃);
 8 |  if Score(x̂) < Score(best_solution) then
 9 |  |  enforce integrality over fractional variables;
10 |  else
11 |  |  best_solution = x̂
12 |  end
13 |  if x̃ has fractional components then
14 |  |  enforce integrality over fractional variables;
15 |  else
16 |  |  STOP = true;
17 |  end
18 |  solve the modified model;
19 |  let x̃ the solution obtained;
20 until STOP;
21 return best_solution;
```

in order to find a feasible solution, while maintaining an anytime behaviour for the algorithm.

We can therefore take advantage of the advanced possibilities offered by some modern LP solvers, in particular the *callback* system. Callback functions are methods that are invoked by a system whenever a certain condition is met, for example the click of the mouse in the browser by the user, or an error that has to be corrected. In our context, we can choose to invoke a method for finding cutting planes whenever a better feasible solution is found, with the *lazy constraint* callbacks. From the geometric point of view, this corresponds to prune the incumbent vertex and move to a (possibly near) vertex

without needing to travel all the polytope. From the optimization point of view, instead, this means to discover a feasible, generally sub-optimal solution and strengthen it little by little, thus raising the lower bound up to the optimal value, in contrast with the original approach that consists in finding an upper bound and lowering it by means of cutting planes. The theoretical justification for this system is the *duality* property of linear programming (section 2.4.1). Figure 4.4 shows a example of how callbacks work, compared with iterative solving.

In practice, it suffices to implement algorithm 4.1 using callbacks, relaxating the model only by removing the acyclicity constraints, without any further linear relaxation.

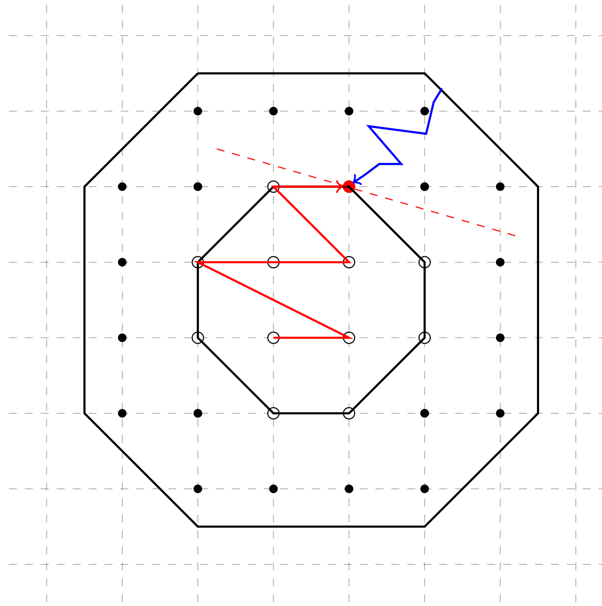


Figure 4.4: geometric effect of lazy constraint callbacks compared with iterative solving in a polytope in $\mathbf{R}^2$. The original polytope is the inner polygone with white integer points, while the outer figure is the relaxated polytope. Callbacks (red arrow) allow to travel directly from an integral feasible solution inside the original polytope to a better one, and cutting planes are used to push forward the objective function. The iterative solving procedure (blue arrow) instead looks for points in the relaxated polytope, and uses cutting planes to reach the optimal solution. The main advantage of callbacks is that the starting point is already known, being the incumbent solution, while the iterative solving has to discover the incumbent solution every time by solving the whole model.

5 Experimental results

In this chapter we show how the approach developed in chapter 4 performs on common test instances. The algorithm has been implemented in the BNSTRUCT package using the R language (R Core Team [88]). The ILP solver of choice is CPLEX 12.5, interfaced using the cplexAPI R package (Gelius-Dietrich [47]).

All the results are reproducible, as the BNSTRUCT package is available at https://github.com/magodellepercussioni/bnstruct under the GNU GPL licence. The dataset have been generated starting from networks found at http://www.bnlearn.com/bnrepository/, and can be found at https://github.com/albertofranzin/data-thesis.

We have tried six instances of different sizes, and for each instance we have generated multiple datasets of 100, 1000 and 10000 items. The instances are ASIA (8 nodes, [71]), CHILD (20 nodes, [105]), INSURANCE (27 nodes, [9]), ALARM (37 nodes, [6]), HEPAR2 (70 nodes, [84]), ANDES (223 nodes, [23]). The first three are “easy” instances, even solvable by complete search approaches such as dynamic programming algorithms. The remaining ones are instead much harder, with also higher in-degree (up to 6 for both HEPAR2 and ANDES).

We have compared the results obtained by our implementation based on CPLEX callbacks (see section 4.6.2) with the results obtained with the GOBNILP package (http://www.cs.york.ac.uk/aig/sw/gobnilp/) configured with CPLEX 12.5 as LP solver, to evaluate the two different approaches that use integer programming, and with the

MMHC algorithm ([120], presented in section 3.3.3 already implemented into BNSTRUCT) in order to compare the two exact approaches against a heuristic.

There are four parameters under evaluation: the score of the final network, the time needed to obtain it, the Structural Hamming Distance between the computed network and the original one (a measure of how much they differ in terms of edges), and the memory allocation requested. As we have multiple instances for each network, we report the median result for each parameter. The tests are executed on a cluster of 14 DELL PowerEdge M600 blades, each with 2 Intel Xeon E5450 CPUs (12MB Cache, 3.00 GHz), 16 GB of RAM. Each instance is allowed to run for no more than 24 hours.

Despite the computational resources exploited, we expect the algorithms to perform well in the easy instances, but also to fail with at least some of the harder instances.

5.1 Preliminary considerations

We have proposed some different approaches in order to solve the model described in chapter 4. However, while theoretically interesting, some of them are not efficient, and, at the current state of our implementation, are not considered for evaluation. For example, the linear relaxation of section 4.6.1 is solved by algorithm4.3 until the optimal solution, and only at that moment the integrality of the solution is evaluated; while, by doing this, the algorithm is able to provide an anytime behaviour, this way is overly cumbersome. Therefore, we evaluate only the callback-based implementation of algorithm 4.1. We also do not look for long cycles in the beginning, as this is not convenient with the implementation tested; we do not provide computational evaluation of this here, but supplementary material will be gradually provided in the dataset repository.

In our implementation, CPLEX is run without the presolve functionalities. Again, we do not show results here as implementation issues with CPLEX are not the focus of this thesis (but results will be provided later on the dataset repository), but, oddly, the presolve does not help at all the solution of the model; indeed, it may heavily slow down the execution. This is very interesting, because the purpose of the presolve is to exploit structures in the model in order to reduce

it, and thus to enable a faster resolution; commercial solvers vendors invest vast efforts in order to develop effective presolving techniques. We do not know why, with our model, the presolve fails so evidently, and we have not tested with other solvers, so we cannot say whether this behaviour is just an unfortunate configuration for CPLEX settings, or there are other reasons.

5.2 Test description

As already mentioned, we evaluate the algorithms over six common synthetic instances, namely ASIA, CHILD, INSURANCE, ALARM, HEPAR2 and ANDES. These instances are chosen to provide a broad spectrum of test cases: the first two are small, "easy" instances, while INSURANCE and ALARM are mid-sized networks, one easier and one more difficult to solve, and the last two are big to very big networks, with also a high in-degree, which raises the difficulty of the task.

For each network, we have generated 20 datasets with 100 items, 20 datasets with 1000 items, and 20 datasets with 10000 items, in order to assess different conditions of data availability. Larger datasets allow the algorithms to behave in a manner that is closer to the ideal case of infinite data, that is, to clearly infer causality relationships. A smaller dataset means instead more difficulty in inferring causality among the variables, and therefore a less efficient work of the statistical tests; furthermore, big datasets are rare in real-world applications, so it is interesting to see how each algorithm may perform in a realistic context.

We choose very standard parameters for our algorithm: the MMPC step has an α threshold for the G^2 test of 0.05, and the scoring function of choice is BDeu with an ESS of 1 (the same function and setting of the other two algorithms tested).

MMHC and GOBNILP are tested with their default settings. The only parameter we set to GOBNILP is the maximum size allowed for the candidate parent set generation, and we use the values reported by the authors in the package homepage, or values reported for similar-sized instances.

As the tests are executed in batch mode, without any interaction, in case of termination by the server for resource exhaustion we cannot retrieve any intermediate result. While our implementation still do

Table 5.1: Real in-degree of the networks used, and in-degree imposed for GOBNILP scoring. For the networks whose results are reported at `http://www.cs.york.ac.uk/aig/sw/gobnilp/` we have used the same limitations; for the other instances, we have chosen a limit that is employed for networks of similar size, such as 3 for HEPAR2 and 2 for ANDES.

Instance	size	# edges	real max in-degree	GOBNILP max in-degree	average degree
ASIA	8	8	2	8	2
CHILD	20	25	2	3	1.25
INSURANCE	27	52	3	3	3.85
ALARM	37	46	4	3	2.49
HEPAR2	70	1236	6	3	3.51
ANDES	223	338	6	2	3.03

not allow this, GOBNILP does, so we cannot evaluate any solution in case of premature termination of a GOBNILP process.

For each instance and size, we compare:

- our implementation of algorithm 4.1 (EO-CP in what follows, short for *edge orientation with candidate parents*) using callbacks, run starting from the skeleton computed with the MMPC algorithm;
- the MMHC algorithm, in order to evaluate how an heuristic approach compares with an exact algorithm over the same starting sparsified skeleton;
- GOBNILP with an upper bound on the cardinality of candidate parent sets (G-Pa), according to each instance (reported in table 5.1);
- GOBNILP over the scores computed with MMPC (G-S); in case of failing of MMPC no results can be provided.

There are two testings that use GOBNILP, in order to clearly show the impact of the MMPC step. The execution of GOBNILP without MMPC is the default one, and the parent set size limitation is the default approach the authors employ, as we have already discussed

in this thesis, and allows a clear evaluation of the two sparsification approaches; conversely, showing how GOBNILP performs over the same set of candidate parent sets of EO-CP and MMHC gives a fair comparison of the quality of the solution algorithms, as they run over the same reduced search space.

We observe:

- the score of the final network;
- the Structural Hamming Distance of the final network, compared with the original network. These two parameters measure the quality of the solution computed;
- the time needed to compute the network, bounded at 24 hours;
- the maximal memory occupation of the process. These two last parameters measure the resources needed by the algorithm.

The Structural Hamming Distance (SHD) between two graphs G and H is defined as the number of transformation via edge insertion, removal or re-orientation (in case of directed graphs) needed to transform G into H. It is therefore a measure of the similarity of the two graphs.

For each of these metrics we report the median value among the 20 results obtained, one for each dataset for every instance and size. When some of the tests have failed to terminate, we report it with the explanation. An instance may fail due to time or memory exhaustion; while the time depends only on the instance, the memory available depends also on the load of the machine, so memory exhaustions may happen at different quantities.

5.3 Results

We now report the results obtained for the metrics analyzed. The tables contain the median value among the results computed for the successful terminations of each trial.

5.3.1 Successful tests

First we show in table 5.2 how many trials have terminated successfully for each instance, and, if the instance has failed, we explain why it happened. The failures may be due to time or memory exhaustion.

Table 5.2: summary of how many trials, among the 20 for each instance, have terminated succesfully.

Instance	# items	EO-CP	MMHC	G-Pa	G-S
ASIA	100	20	20	20	20
	1000	20	20	20	20
	10000	20	20	20	20
CHILD	100	20	20	20	20
	1000	20	20	20	20
	10000	20	20	20	20
INSURANCE	100	20	20	20	20
	1000	20	20	20	20
	10000	20	20	20	20
ALARM	100	-	20	20	20
	1000	19	20	20	20
	10000	20	20	20	20
HEPAR2	100	17	20	20	19
	1000	20	20	20	20
	10000	20	20	20	20
ANDES	100	-	20	-	-
	1000	19	19	-	19
	10000	20	20	8	20

We consider a success the discovery and certification of the optimal solution (or a local one, for MMHC). In some cases the algorithm may have determined a solution, but failed to certify it as the optimum. Due to the structure of the tests, this is considered a failure.

This, however, may be a too strict evaluation for a practical context: GOBNILP has an anytime behaviour, and could still provide the incumbent solution if stopped during the execution. EO-CP has also a theoretically anytime behaviour, though the current implementation does not allow to return an intermediate solution.

As it is clear from table 5.2 small instances are not an issue for any of the algorithms. As the size grows, while MMHC still terminates without problems (except one case for ANDES_1000), exact approaches instead begin to encounter some failures. The problems

Figure 5.1: failure of scoring for HEPAR2_100.

```
[1] "- evaluating node 52"
(has 45 possible parents, and therefore up to 3.518437e+13
  candidate parent sets to evaluate)
[1] 1
5 / 45
[1] 2
3 / 990
[1] 3
1 / 14190
[1] 4
1 / 148995
[1] 5
1 / 1221759
[1] 6
1 / 8145060
[1] 7
1 / 45379620
[1] 8
1 / 215553195
[1] 9
1 / 886163135
[1] 10
```

of EO-CP are due to running out of time. For ALARM_100, in 7 trials the execution has been terminated during the computation of the scores for the CPCs, while in the remaining ones it was the edge orientation part that took too long and was aborted. One instance of ALARM_1000 also failed, and was terminated when CPLEX got a gap among the bounds of 0.42% (having therefore a good candidate solution as incumbent, probably the optimal one, but failed to certify it). There were 3 failures for HEPAR2_100, whose scoring step got stuck on a node with too many neighbours; in figure 5.1 we show what happened in the scoring step in those cases, where a single subset got expanded, causing the evaluation of too many other unnecessary par-

ent sets. As for ANDES_100, all of the trials got aborted during the scoring step; the one instance of ANDES_1000 that failed, instead, got terminated during the MMPC step.

GOBNILP instead always returned a solution for all of the datasets but ANDES. All of the trials for datasets of 100 and 1000 items, and the 12 trials with 10000 items that terminated in advance, got aborted due to excessive memory demand. Such high requirements were caused by the development of the branching tree over a high number of variables (several thousands), despite the fact that the candidate parent sets were imposed a maximum size of 2.

Failures for GOBNILP when run over the scores computed after the MMPC step are all due to abortion of the scoring step or the MMPC one, as GOBNILP, otherwise, always returned the optimal solutions in few instances. Results in terms of score of this setup may differ from EO-CP because of possible failures of the CPLEX-based part of EO-CP (thus varying the median value); results in terms of SHD may vary for both failures of EO-CP and the fact that there may exist multiple solutions with the same score, and GOBNILP and EO-CP return different solutions of the same score, but with a different edge combination.

5.3.2 Score of the networks

As it is clear from table 5.3, GOBNILP with the limitation on parent set size is the choice that leads to the best scoring networks for smaller networks, while EO-CP (and of course GOBNILP ran over the scores computed after MMPC) usually comes second best. This is to be expected, since when an algorithm evaluates candidate parent sets of size at least equal the degree of the original network, the search is, in practice, complete. The MMPC step, instead, may discard some true parents and therefore some optimal configurations, resulting in lower-scoring networks.

As the size of network grows, and so does the maximum in-degree, while GOBNILP remains the solution that yields the best scores, the gap with EO-CP narrows. It is interesting that, even if the maximum in-degree allowed to GOBNILP is lower than the real one, it still provides better-scoring networks in many cases. This is probably due to the fact that, while the loss of prospective candidate parent sets of large size entail a loss of information, and therefore of score,

Table 5.3: summary of score (BDeu, ESS=1) results, rounded to the nearest decimal. The higher the better; the best results for each instance are highlighted in bold.

Instance	# items	EO-CP	MMHC	G-Pa	G-S
ASIA	100	−262.2	−262.3	**−243.1**	−262.2
	1000	−2384.7	−2384.7	**−2280.7**	−2384.7
	10000	−24 046.3	−24 046.3	**−22416.8**	−24 046.3
CHILD	100	−1472.0	−1483.6	**−1466.7**	−1472.0
	1000	−12 848.5	−12 848.5	**−12848.5**	−12 848.5
	10000	**−122900.4**	−122 977.6	**−122900.4**	**−122900.4**
INSURANCE	100	−1656.4	−1666.3	**−1652.5**	−1656.4
	1000	−14 097.3	−14 193.4	**−14019.2**	−14 097.3
	10000	−132 795.9	−133 138.1	**−132562.1**	−132 795.9
ALARM	100	-	−1279.7	**−1241.2**	−1241.9
	1000	−10 662.9	−10 806.4	**−10589.7**	−10 656.9
	10000	−101 160.2	−108 917.3	−101 160.2	**−101065.5**
HEPAR2	100	−3425.5	−3451.5	**−3329.5**	−3435.6
	1000	−33 190.4	−33 200.7	**−33153.3**	−33 190.4
	10000	−326 636.2	−326 769.1	**−326420.2**	−326 633.8
ANDES	100	-	−10 764.8	-	-
	1000	**−98273.7**	−99 476.8	-	**−98273.7**
	10000	−952 502.4	−959 474.9	**−944432.3**	−952 502.4

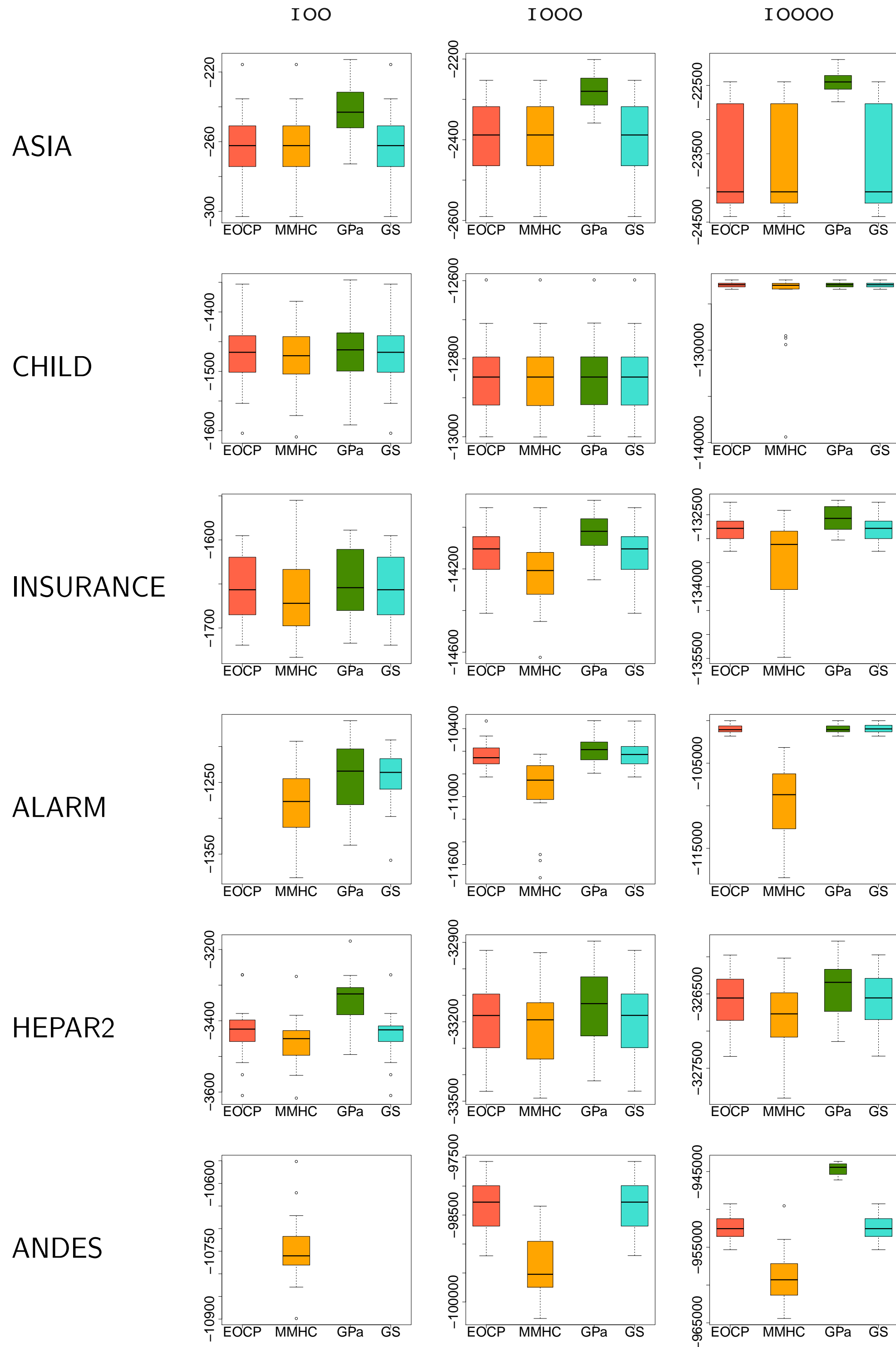


Figure 5.2: score results for, left to right in each picture, EO-CP, MMHC, G-Pa, G-S.

this gets amortized by a more complete coverage of the variables with fewer parent sets, especially if also EO-CP fails to identify correctly the bigger parent sets (as is the case with HEPAR2 and ANDES). The errors due to statistical inference show how a conservative test may be harmful for small instances.

It is also clear how the amount of observed data impacts over the algorithms based on MMPC: few data yield much unreliable results, while, as the size of the dataset grows, their performance gets closer to GOBNILP. GOBNILP, instead, does not perform any statistical pruning before the executions, and is therefore less susceptible to the bias that a dataset has when composed of few items.

5.3.3 Structural Hamming Distance

In table 5.4 we report the results regarding the Structural Hamming Distance. It is interesting that the table does not reflect the situation that we obtained for the scores, but instead it shows that EO-CP algorithm is more performing than its competitors, often yielding better results than GOBNILP. Especially when the size and complexity of networks grow, the approaches that evaluate (possibly) larger parent sets according to statistical tests, and therefore according to the data, significantly outperform GOBNILP for this metric. For ANDES_10000 the approaches based on MMPC yield up to roughly half of the errors with respect to the networks discovered by GOBNILP with the parent set limitation.

Interesting, with smaller datasets the best results are often obtained with MMHC, the only approximate algorithm of the lot. With larger datasets, instead, exact approaches have lower SHD, as expected since they are *sound* (see section 3.3).

The discrepancy among tables 5.3 and 5.4 suggests that the scoring function may suffer of overfitting issues when the ratio among items in the dataset and the number of variables in the graph is low. This issue can be addressed by tuning the ESS in the scoring function, hence by assuming a different confidence in the prior knowledge.

5.3.4 Time performances

In table 5.5 we report the timing of the algorithms for the instances that terminated successfully. Instances that got terminated after 24

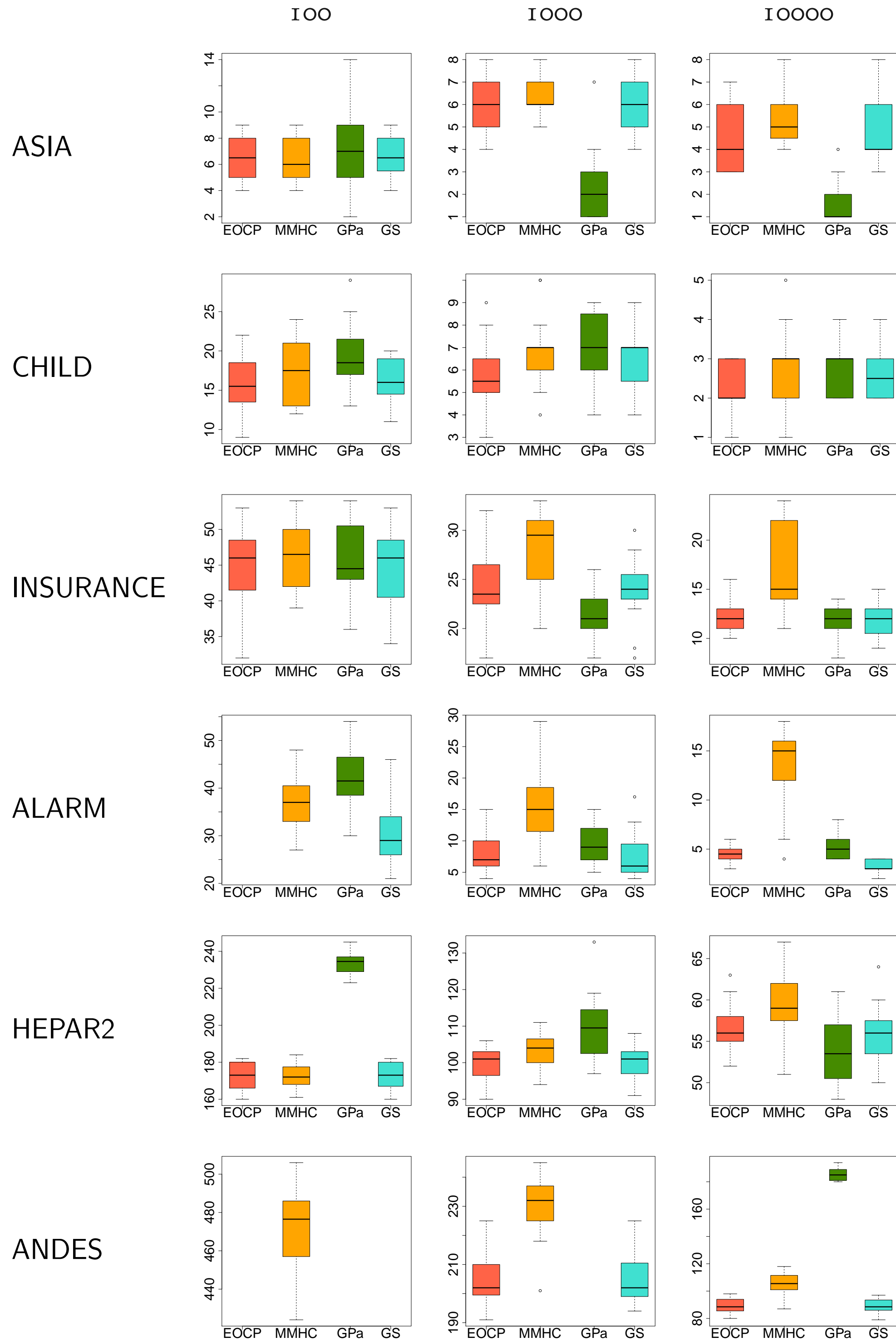


Figure 5.3: SHD results for, left to right in each picture, EO-CP, MMHC, G-Pa, G-S.

Table 5.4: summary of Structural Hamming Distance results with respect to the original network used to generate the data. The lower the better; the best results for each instance are highlighted in bold.

Instance	# items	EO-CP	MMHC	G-Pa	G-S
ASIA	100	6.5	**6**	7	6.5
	1000	6	6	**2**	6
	10000	4	5	**1**	4
CHILD	100	**16**	18	19	**16**
	1000	**5**	7	7	7
	10000	**2**	3	3	**2**
INSURANCE	100	46	47	**45**	47
	1000	23	29	**21**	24
	10000	**12**	15	**12**	**12**
ALARM	100	-	38	42	**30**
	1000	7	16	9	**6**
	10000	4	15	5	**3**
HEPAR2	100	173	**171**	234	172.5
	1000	**101**	104	109	**101**
	10000	56	59	**54**	56
ANDES	100	-	**476**	-	-
	1000	**202.5**	232	-	204.5
	10000	**88**	106	185	**88**

hours are therefore not counted. For each algorithm we show the time spent in the preprocessing (including the MMPC step when used), the time spent in the actual edge orientation or model solving, and the total time of the execution of the whole process as measured by the UNIX `time` utility.

With small instances, the lenghty part of EO-CP is the MMPC step, while the actual CPLEX part takes very little time. It turns out that ALARM is a hard instance for its size, and EO-CP takes significantly more time than its competitors, especially for small datasets, where no trial terminated on time. For larger networks, if the preprocessing is effective the edge orientation takes very little time; the times for ANDES are the least among the algorithms tested.

Table 5.5: summary of time needed for solving the instances, the lower the better.

Instance	# items	EO-CP Tp [s]	EO-CP Ts [s]	EO-CP Tt [s]	MMHC Tp [s]	MMHC Ts [s]	MMHC Tt [s]	G-Pa Tp [s]	G-Pa Ts [s]	G-Pa Tt [s]	G-S Tp [s]	G-S Ts [s]	G-S Tt [s]
ASIA	100	0.776	0.01	0.784	0.055	0.015	0.072	0.04	0.024	0.064	0.786	0.037	0.823
	1000	0.794	0.01	0.809	0.057	0.023	0.081	0.07	0.038	0.11	0.804	0.04	0.844
	10000	0.911	0.01	0.918	0.083	0.047	0.124	0.24	0.149	0.387	0.921	0.041	0.962
CHILD	100	2.462	0.13	2.572	0.135	0.204	0.343	0.05	0.16	0.207	2.522	0.048	2.57
	1000	2.309	0.25	2.564	0.464	0.215	0.682	0.13	0.322	0.449	2.369	0.039	2.408
	10000	12.243	0.56	12.812	9.731	0.4	10.182	0.9	1.435	2.33	12.303	0.043	12.313
INSURANCE	100	28.138	15.53	81.189	0.462	0.871	1.324	0.27	0.434	0.722	28.298	0.044	28.342
	1000	2.126	2.18	4.499	0.864	0.544	1.43	1.01	0.97	2.009	2.266	0.041	2.307
	10000	13.932	2.23	16.725	11.548	1.233	13.148	8.63	4.72	13.323	14.212	0.049	14.261
ALARM	100	-	-	-	1.588	2.502	3.925	5.13	1.135	6.331	15 298.1	0.219	15 298.319
	1000	6.38	9001.205	9013.454	2.122	1.133	3.231	5.89	2.801	8.911	7.24	0.062	7.302
	10000	32.076	1305.42	1335.328	25.884	1.584	27.27	704.62	15.867	720.318	34.006	0.064	34.07
HEPAR2	100	134.862	1.71	137.092	13.95	10.065	24.071	586.77	10.314	597.212	134.942	0.0455	134.987
	1000	720.55	0.24	720.98	1112.122	3.782	1117.194	0.66	26.883	27.58	720.63	0.045	720.675
	10000	241.921	1.22	243.391	225.041	5.228	230.205	3.83	184.197	188.276	242.071	0.071	242.142
ANDES	100	-	-	-	839.985	284.848	1168.035	-	-	-	-	-	-
	1000	2023.26	1.72	2024.495	2042.341	185.7535	2237.0495	-	-	-	2025.855	0.061	2025.916
	10000	234.157	2.44	236.297	237.165	241.448	474.794	26 180.43	254.817	20 496.397	234.687	0.125	234.812

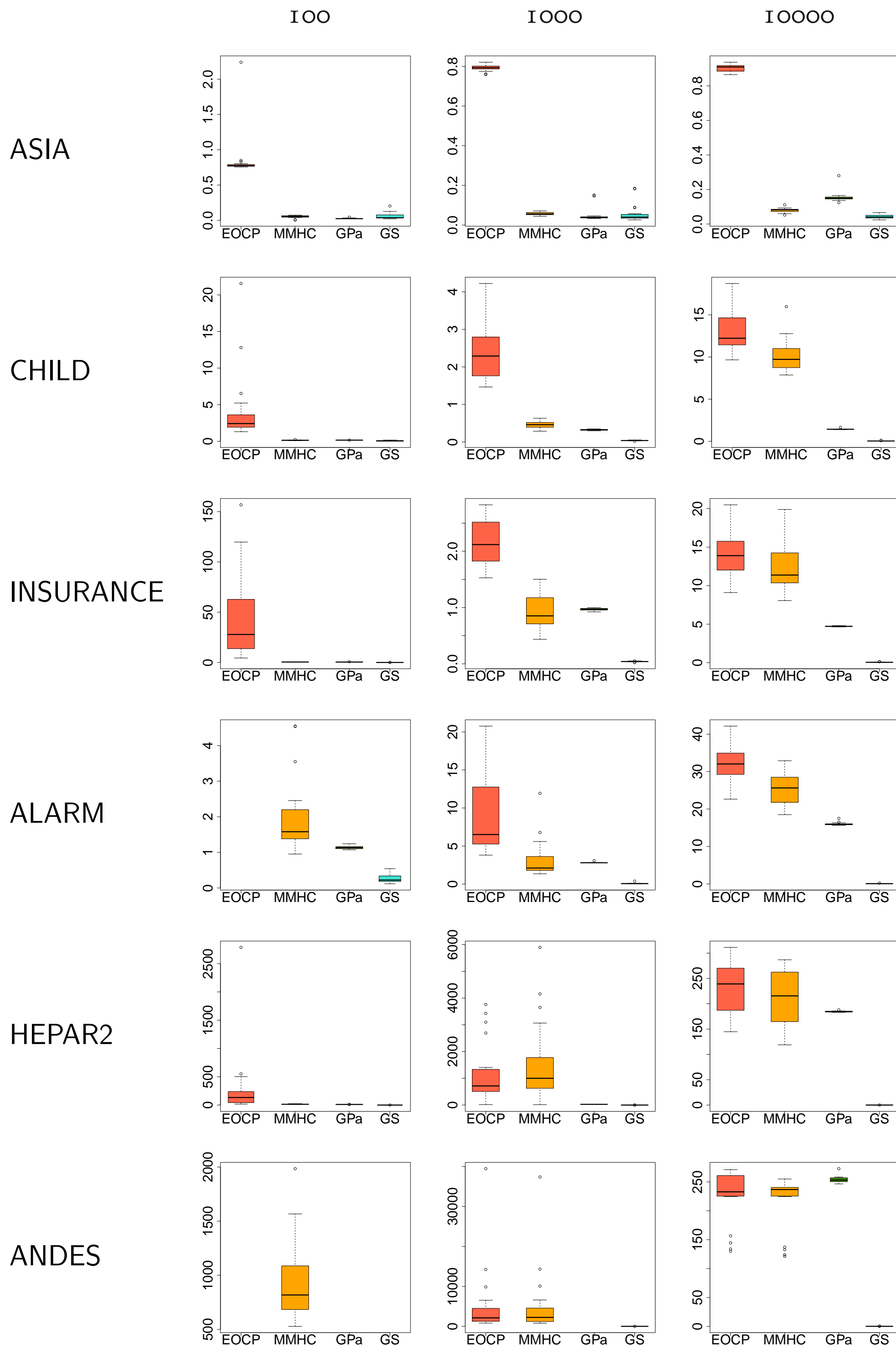


FIGURE 5.4: preprocessing time results for, left to right in each picture, EO-CP, MMHC, G-Pa, G-S.

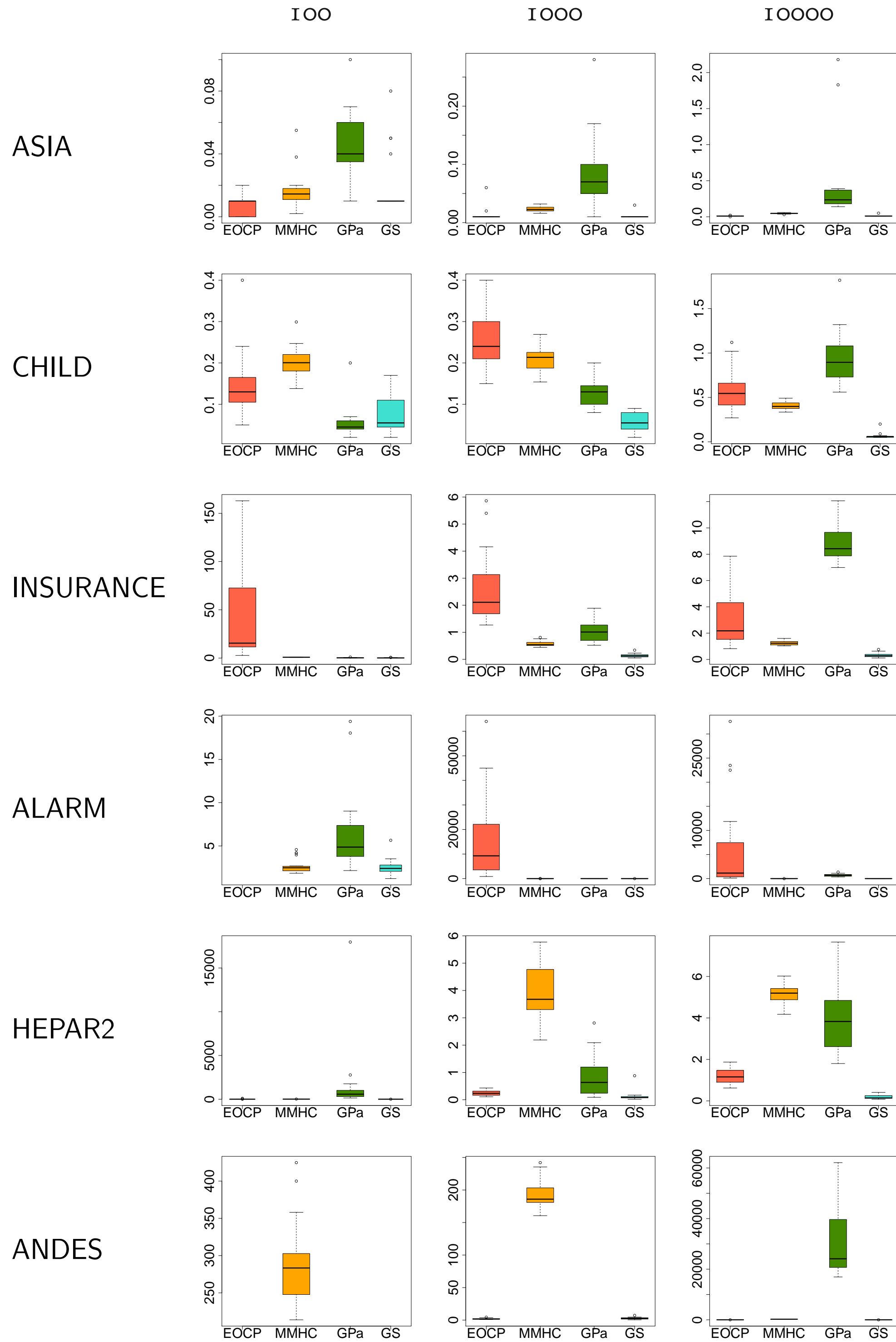


Figure 5.5: edge orientation time results for, left to right in each picture, EO-CP, MMHC, G-Pa, G-S.

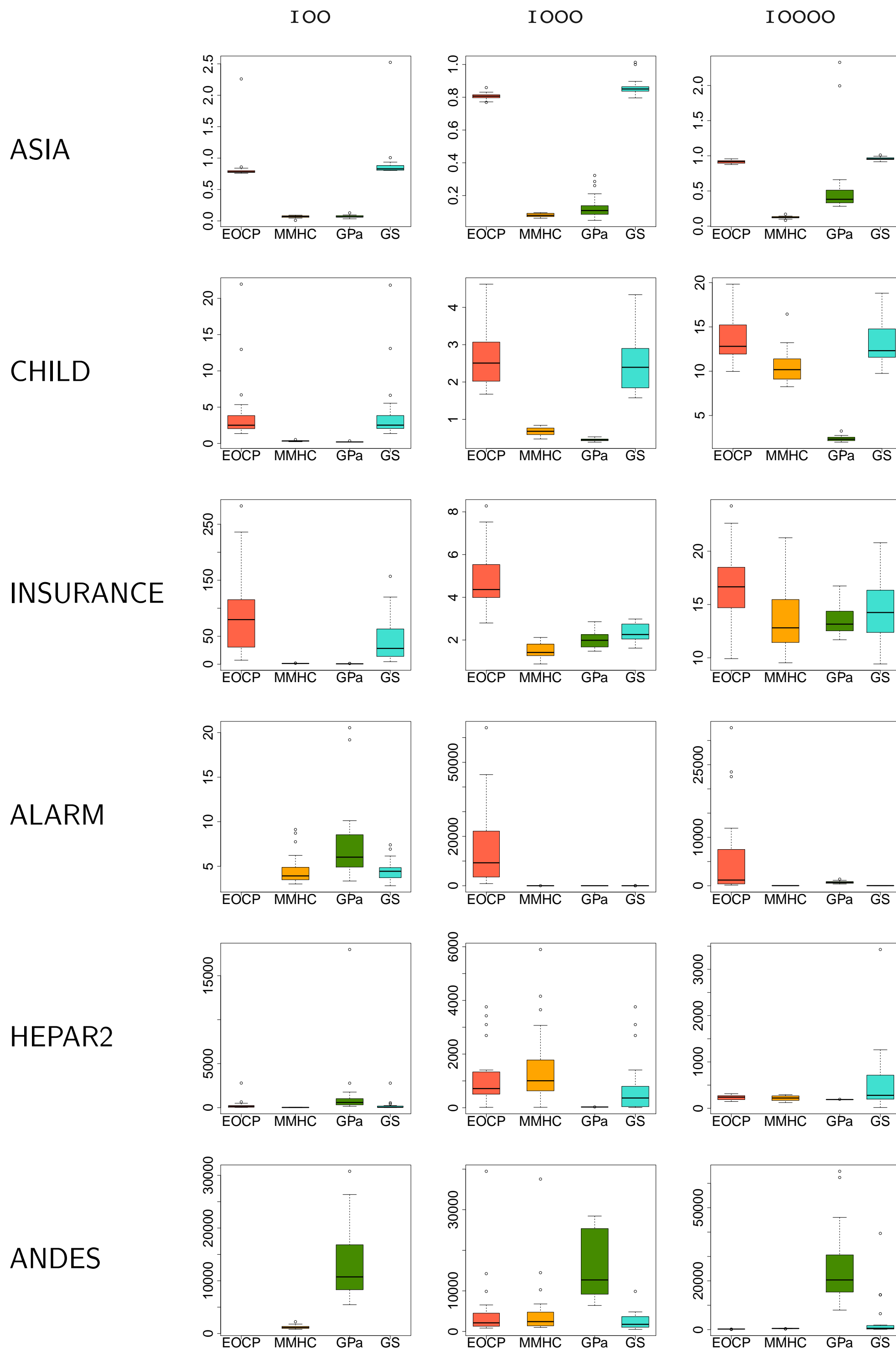


Figure 5.6: overall time results for, left to right in each picture, EO-CP, MMHC, G-Pa, G-S.

The speed of the preprocessing is related to the size of the datasets: a more effective statistical test yields a slower MMPC step, but the sparser networks it entails allows a much faster scoring of candidate parent sets, which is a true bottleneck for the tests with 100 items. There are some discrepancies among the values for EO-CP and MMHC: for example, for some large networks the preprocessing for EO-CP takes some noticeable less time than the MMCP for MMHC, which is a bit odd. A possible explanation for this lies in the fact that the load on the shared machine employed for the tests may have slown down some tests; as the instances were ran in parallel, the same delay impacted many of the trials.

GOBNILP with limitation on CPC size is instead much faster for small instances. Conversely, as the number of nodes in the network grows, the bound over the CPC size does not prevent the insertion in the model of a consistent amount of variables that heavily burden the resolution. For the ANDES network there are $223 \times 223 \times \binom{223}{2} \approx 1,23 \times 10^9$ candidate parent sets to evaluate, a process that takes several hours. The model solving is also quite time consuming, due to the many variables in the model.

More emphasis on the impact of the number of variables over the computational resources requested is given by the comparison with the behaviour of GOBNILP when fed with the scores precomputed after MMPC: with the exception of ANDES_100, for which the MMPC+scoring step exceeds the 24 hours limit, it takes instants to solve every instance, even the largest ones. This is indeed a corroboration of the goodness of the model employed by Cussens, since it runs even faster than the HC heuristic, with exact results.

5.3.5 Memory allocation

Table 5.6 contains the median values for the memory demand of the algorithms. Missing values (except for G-S for ANDES_100 where the algorithm was not ran) indicate that the execution terminated so quickly that it was not possible to measure the memory usage, and may therefore be viewed as the less demanding solutions.

The MMPC step and the HC algorithm have low memory requirements, almost constant with respect to the size of the instance. The solutions based on LP solving, instead, require a higher amount

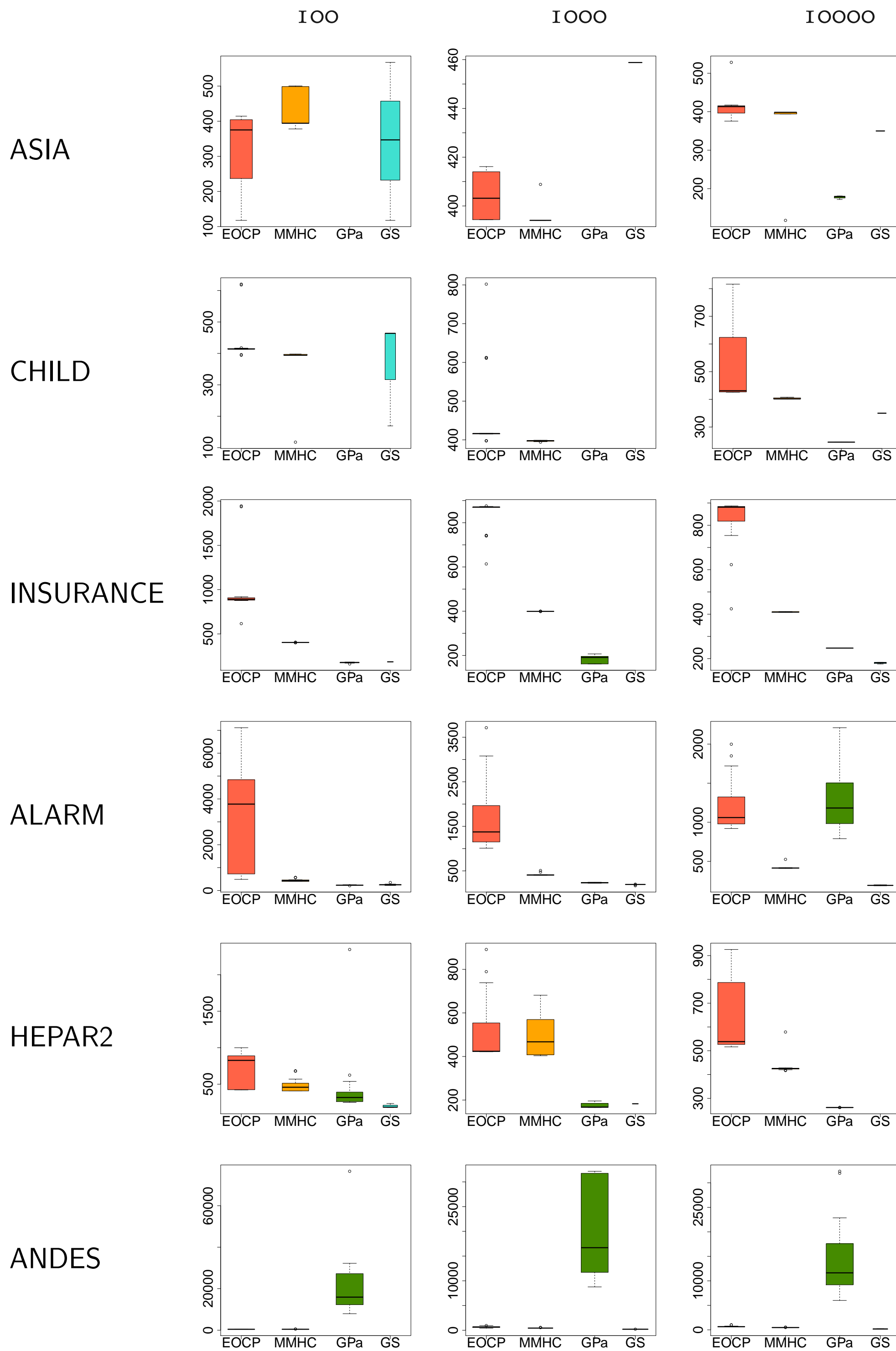


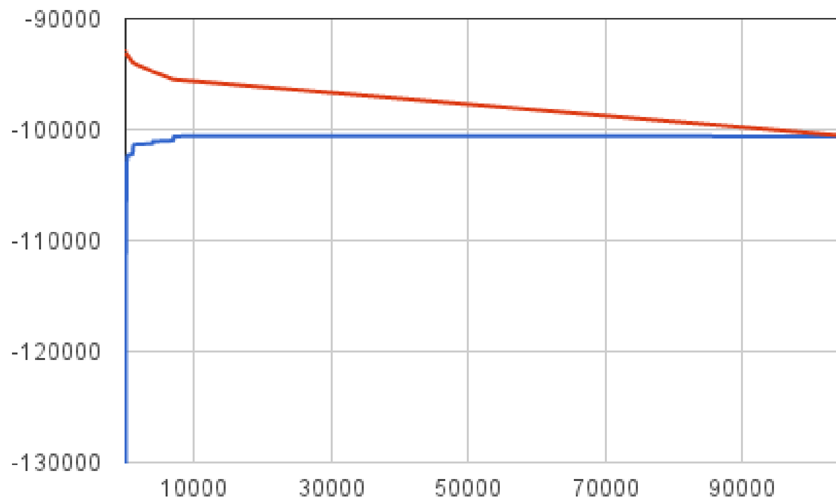
Figure 5.7: memory results for, left to right in each picture, EO-CP, MMHC, G-Pa, G-S.

Table 5.6: summary of memory requirements for the instances, rounded to the nearest integer. The lower the better; the best results for each instance are highlighted in bold.

Instance	# items	EO-CP	MMHC	G-Pa	G-S
ASIA	100	375	394	-	347
	1000	412	394	-	459
	10000	413	398	**180**	350
CHILD	100	414	395	-	464
	1000	416	398	-	-
	10000	431	402	**245**	350
INSURANCE	100	894	404	**179**	185
	1000	871	399	192	-
	10000	880	410	2480	**182**
ALARM	100	3906	403	**231**	250
	1000	1384	408	234	**196**
	10000	1071	414	1187	**192**
HEPAR2	100	824	458	310	**184**
	1000	425	467	**166**	183
	10000	538	425	262	-
ANDES	100	426	438	16 481	-
	1000	639	438	16 941	**204**
	10000	636	475	11 884	**211**

of memory to store the entire branching tree they employ. A small number of variables is a key factor in maintaining low memory requirement. The growth of the branching tree, especially in a structure as the one of CPLEX, which traverses it in order to find the most promising nodes to evaluate, implies that a much larger amount of memory is needed, and that more computational time is spent traversing it, without actually *solving* the instance. This is one of the points where the limits of the maximum candidate parent set size shows up: many variables are kept in the model without a real need, burdening the resolution, as is shown by the different computational results of ANDES_10000, where the CPLEX part of EO-CP terminated in few seconds, while its counterpart in GOBNILP sometimes failed to ter-

Figure 5.8: Evolution of score of the ALARM_10000 network. The blue line is the primal (lower) bound, and we can see it quickly converges to the optimal value, while the red line is the dual (upper) bound, which converges much slowly.



minate, getting aborted by the scheduler for memory exhaustion, in some cases having allocated more than 30 GB of RAM. The performances of GOBNILP when launched over the candidate parent sets computed by MMPC strengthen this observation, with the algorithm that terminates in few instants, allocating an almost negligible amount of memory.

5.4 Comments

First we briefly report (figure 5.8) an example of how a problem is solved via cutting planes with EO-CP, showing how the upper and lower bounds converge to the optimal value. Since GOBNILP also uses CPLEX, it exhibits an analoguous evolution. We report the picture for ALARM_10000 as an example of the behaviour in a situation of lengthy computation. For small instances the problem can be solved at the root node of the branching tree.

In case of a rapid solution the gap closes very quickly from both the upper and the lower side. When the computation needs more

time, instead, we note how the optimal solution is indeed discovered quite quickly, after "few" iterations. It is a very common situation among the instances, while it may happen in some instances that the optimal solution is computed in late iterations; in this case, however, generally the major updates of the lower bound happen in the beginning of the iterations, and later updates are just refinements of the incumbent value. Mainly, what really takes a lot of time is to lower the upper bound, in order to close the gap from above. Even when the instance has been terminated for having exceeded the time limit, while we cannot confirm the optimality of the incumbent, we can see that the lower bound has not improved, a sign of the fact that in the last millions of nodes of the branching tree processed there was no better feasible solution; therefore, if the solution in hand was not the optimal one, it was al least a very good one.

The really difficult task is therefore to enhance the upper bound. It is in fact a normal situation when solving integer programming problems, where a good dual bound is often more important than a very good primal bound[1].

EO-CP has proved to scale well, and has even solved with good results all of the instances of ANDES with datasets containing 1000 and 10000 items. This is largely due to the effectiveness of the MMPC step, that heavily sparsifies the graph and allows the model to be composed of few variables, and therefore few constraints. By contrast, with few data, the statistical test performs poorly, and, in the given time, the algorithm has failed even to compute all of the scores for the variables. It has also to be reported that some instances, such as ALARM, which are hard for their size, can require significant time to complete. Furthermore, time performances may vary a lot, heavily depending on the observed dataset. Smaller instances can be instead solved even at the root node of the branching tree.

GOBNILP instead performs better on the instances with smaller nodes and low in-degree. In particular, when the "hypothetical" in-degree is equal or higher than the real one, the search is effectively done in a neighbourhood of the optimal solution. For example, GOBNILP is applied by the authors for pedigree reconstruction [29] where the maximum in-degree is two: in that case, this approach is perfectly

[1] In fact, in minimization problems where costs or scores are positive, it is the lower bound that requires a long time to reach the optimal value.

sensible, as prior knowledge over the domain ensures the optimal network to be found. This approach, however, proves to be expensive when scaled up to larger networks: the number of variables in the model grows up to several thousand for ANDES, resulting in a cumbersome branching tree, burdened by a vast amount of unnecessary variables. Nonetheless, the scores obtained by GOBNILP— when it terminates — are slightly higher than the scores of the networks computed by EO-CP, even if the networks cannot have nodes with more than two parents.

The fairest comparison among the two approaches is given by the results of GOBNILP when fed with the scores computed by the MMPC step, letting it run over the same variables that EO-CP runs on. In all of the cases, GOBNILP terminates almost immediately, so fast that the server in many trials cannot even measure the memory allocation. As it is clear from the results provided in this chapter, running the MMPC algorithm as first step of a structure learning instance allows the power of the model underlying GOBNILP to be fully exploited, resulting in comparable or better quality results while, in some cases drastically, reducing the computational demandings.

We can give two main explanations for the difference of performance between the two models. The first one is that as violated cluster constraints are searched in the fractional solution, infeasible solutions can be ruled out at every node of the branching tree; furthermore, the quest for the most promising cuts helps in keeping the model as light, hence faster to solve, as possible. The acyclicity constraints defined over the edge indicator variables are instead evaluated only when an integral solution is found, a fact that lets the branching tree grow without any external control, possibly analyzing many invalid solutions before thay are effectively recognized as cyclic digraphs.

The second main reason is that, by letting the edge indicator variables and the bounding constraints that connect them to the CPC variables out of the model, the model is smaller and less constrained, and therefore easier to solve. The model used by Cussens may be seen as a restricted version of the more general model we have developed in this work. All of the information needed to represent the instance is, of course, contained in both models, but while our model features both the CPC point of view and the edge point of view and connects them, the model provided by Cussens only shows the CPC side. The simplification in the model entails however a more difficult separation

problem since it is needed to infer the graph and manage acyclicity constraints using only the CPC variables. However, we have seen this is not an issue from a practical point of view, at least for the testbed we have used in this work.

We have maintained the same setting of the MMPC algorithm for all the tests, but for very small networks it is clear that a complete search can be done over all of the possible candidate parent sets, as the MMPC step offers no practical avdantage in terms of computational requirements to compensate the loss of information of the pruning step. It is also clear from the reported results that when few observations are available a statistical pruning of the space may be problematic. In case of scarce data the G^2 test cannot correctly infer the connections among the variables, and is forced to keep too many configurations, as there is no evidence for them to be irrelevant; the generation of the score becomes, in this case, too lengthy to terminate. For larger datasets, instead, the data-based pruning is more performing, and effectively yields a much lower number of variables with respect to the competing approach. On the other hand, the test may incorrectly discard nodes belonging to the optimal configuration, and therefore yield a worse network. As happened with ANDES_10000, a network with original maximum in-degree of 6, the MMPC step followed by the BDeu scoring has yield a network with maximum in-degree never higher than 4; GOBNILP, performing a full search over the candidate parent sets with maximum cardinality of 2, has discovered networks with higher score.

However, the results for the Structural Hamming Distance show a different picture, with the approaches performing MMPC that yield networks more similar to the original ones. This is a clear indication of how a statistical sieving of the search space may result in networks more adherent to the real ones than networks discovered after arbitrary search space pruning; we can therefore say that the MMPC step definitely helps us in nearing our final goal. The real effectiveness of MMPC is however subject to the amount of data at hand. With few data the statistical inference gives unreliable outcomes, while with lots of data the MMPC step may be a bottleneck for the computation, and it has happened that some instances have not completed the MMPC step in the 24 hours given: the amount of data contributes not only to the quality of the solution, but also to the time MMPC takes to complete.

We have compared two integer programming approaches, with two different ideas of how to cope with the combinatorial explosion of the number of variables. Looking at the computational results, for non-trivial networks it seems reasonable to perform statistical testing when there is enough data to do so; to avoid degenerate cases of CPC scoring some limitations over the number of parents may be imposed, but this limitation can be set at a more permissive, and therefore less arbitrary, level than what is done in the original works we have studied in section 3.3.6. An even better setting consists in an "adaptive" limitation, based on what effectively happens during the scoring step (see for example figure 5.1: in this case we could decide to stop at a certain level, while ignoring larger candidate parent sets). As the solutions are however efficient, especially the method used in GOBNILP, the tests can be made more permissive, with a looser threshold, in order to prevent as much as possible the exclusion of a real connection among the variables, thus avoiding to discard the optimal network.

When the data is scarce, the statistical tests may even fail to obtain some results, and therefore a blind (data-independent) pruning may still be the more viable choice, since the loss of information it entails is matched by the loss of precision, paired with supplementary work requested, of the statistical tests.

Conclusions 6

In this thesis we have studied the problem of learning a Bayesian Network structure from a dataset. We have initially described the Bayesian Networks paradigm and properties, accompanying it with a brief but complete overview of the basic graph theory, probability theory and integer programming concepts needed to understand the remainder of this work.

We have then summarized the state-of-the-art works in the literature about the problem of Bayesian Network structure learning, explaining the main approaches followed, with a much deeper focus on scoring-based methods. We have described the most common scoring functions used, and have proposed a broad, but still partial, overview of the principal existing algorithms for the task. As there are very many algorithms for structure learning, we have been forced to leave out some of them, reviewing only the most popular ones. In particular, we have reviewed some recent works based on integer programming, in which the authors formulate the structure learning problem as mixed integer programming model and employ optimization methods such as gradient descent and branch-and-cut over some relaxation of the model, in order to solve it in a more efficient way.

As the IP formulations have exponential size, some limitations have to be imposed on them, in order to cope with medium-to-large instances. The authors of those works choose to assume a maximum cardinality for candidate parent sets, in order to stop the combina-

torial explosion entailed by the growth of the number of observed variables. We have discussed how this limitation is arbitrary and not based on the observed data, yet it still leads to an unnecessary amount of variables and constraints in the model, and moved from this considerations in order to develop an alternative solution.

We have therefore borrowed the idea of performing an earlier statistical sparsification of the search space by discovering a directed graph called skeleton, which underlies the final DAG we have to find. While the idea is absolutely general, we perform this statistical test using the MMPC algorithm. With the skeleton in hand, we can restrict the scoring, and therefore the number of variables in the IP model, to a set of candidates that is both smaller and related to the dataset, overcoming in most cases the shortcomings of the arbitrary limitation over the size of CPCs. We have also extended the IP model of some of the previous works, including the notion of edges and connecting them to the candidate parent sets, maintaining the linearity property of the model.

We have also discussed how the model we have proposed connects the notion of candidate parent sets with the edges of the DAG, enabling two different points of view on the problem; our model can be therefore be seen as underlying many of the existing approaches, which usually address only one of the two aspects of structure learning. We have also suggested some connections with other problems, such as the generalized version of the more studied Traveling Salesman Problem.

We have implemented our solution in the BNSTRUCT package, using CPLEX as LP solver. We have compared our solution against the GOBNILP package, another IP-based solver, and the MMHC heuristic. Computational evaluations show that our solution can be competitive, and for some larger networks even superior, to the default settings of GOBNILP and MMHC. The MMHC heuristic is fast but generally yields to the worse networks, even if with few data it is still very competitive. GOBNILP is very good for smaller instances, but for large instances its sparsification strategy is not very effective, and is very resource-demanding. The performances of our algorithm depend on the results of the MMPC step, and therefore on the amount of observed data. When the datasets have limited size, the statistical evaluation is less effective, and therefore it yields worse results. When, instead, the amount of data allows some valid sta-

tistical inference, our solution shows encouraging results even for the largest instances of our testset. Quality results show mixed behaviour, with GOBNILP usually discovering better-scoring networks, but with larger distance with respect to the original networks. We have also hybridized our approach by feeding GOBNILP with the outcomes of the MMPC algorithm: this is probably the winning approach, as the quality of the IP-based solution it implements can be better exploited when the model is restricted to a set of candidates more adherent to the real ones.

Our idea of restricting the search space employing statistical tests is therefore an effective approach, viable both *per se* paired with our model and when embedded into existing frameworks, as we have proven by making GOBNILP return better results than when run in its original setting.

6.1 Future directions

As this thesis is based on a novel approach, there are plenty of possibilities for further studies. First, one possibility is to deepen the chance of improving the existing framework by proper parameter tuning and code optimization. For example, we have already planned to analyze the behaviour of the algorithm using different equivalent sample sizes, different scoring methods, and difference confidence thresholds in the independence tests; moreover, it seems that the separation procedures can be further enhanced, perhaps by considering different strategies such as clique discovery. This will, in general, add more cuts at each step, and it may avoid to consider some infeasible solutions.

The solution implemented using the callbacks is theoretically anytime, but the implementation currently do not allow this behaviour to be exploited, so this is surely a feature to add. A more general implementation of the solver, which can use open MIP solvers such as COIN-OR or GLPK, may also help in making this approach available for public use.

Another improvement concerns the integration of the existing approaches into our framework, for example evaluating the impact of discovering violated cluster constraints in the fractional solution at the root of the branch-and-cut tree, and possibly in other nodes; as this means to look for violated constraints in both the fractional and the

integer solutions, a good balancing between the two techniques may yield better performances. Furthermore, another possible direction is to study the possibility of integrate the characteristic imset into the model: while Lindner [73] show a connection to the model proposed in Jaakkola et al. [57], the introduction of edge indicator variables, and their relation with candidate parent set indicator variables, may enlighten new relationship between the two approaches.

The implementations provided so far heavily rely on the "good behaviour" of a default CPLEX setting. While this eases the work of the developer, it also leaves him/her exposed to the impredictable erraticity of LP solvers. For example, in our setting the CPLEX presolve algorithms heavily slow down the execution, which definitely is an odd situation whose causes are to be studied. A custom branch-and-cut strategy, which employs a full battery of cuts at each node, is therefore a path to explore.

While the previous approaches consider the "full" structure learning problem as an integer program, we have employed a mixed strategy, limiting the application of MIP techniques to a set of CPCs reduced using statistical methods (the model is, however, still valid even for skeletons composed of complete graphs). However, the MMPC step for highly interconnected structures can be a bottleneck. Another possible development is therefore to study if mathematical programming techniques can be applied also in the context of skeleton discovery. A parallel pathway is to consider alternative methods to MMPC among the solutions already in the literature, such as algorithms for Markov Blanket discovery.

We have also observed how scoring function outcomes can be misleading, having better-scoring networks with a much larger distance from the original network with respect to lower-scoring networks, even with large datasets. This is a clear problem, and it is surely to be studied the possibility of improving the existing scoring approaches, of developing alternative scoring methods, or even alternative approaches to the problem.

In the existing literature on Bayesian Networks structure learning, few algorithms that consider Dynamic Bayesian Networks exist. It is straightforward to extend our ILP model to the case of DNB: it just suffices to consider a larger graph, with a higher number of nodes and some edges already forbidden, namely the ones going from a level j to a previous level $i < j$. However, a naive attempt to directly solve

such model is clearly infeasible even for very small networks. A future research of great interest is to develop clever techniques, or alternative formulations, to enable the possibility of tackling such instances.

In existing IP formulations there is also no room for latent nodes, and an extension of the model that considers hidden nodes may therefore be of interest.

We solve the reconstruction step in exact form, meaning that the loss of information is due to the statistical tests in the skeleton discovery. Apart from the already mentioned exploration of different ways for the task, we may try to apply approximate algorithms or heuristic methods also for the edge orientation step, maybe starting from a feasible but suboptimal network previously retrieved in some way. This may help especially in case of huge networks to find.

So far, we have only provided many possible future directions only considering the existing theory of the IP approaches for structure learning. The field of mathematical programming is, however, very rich of advances techniques whose applicability on this problem is up to study, both for theory and practice. Graph theory may also have untested tools, and the new connection between edges and candidate parent sets may suggest to apply some other existing method. As we have argued our model to be, a posteriori, at the base of many existing approaches, it is likely that it can be used to study new improvements to previously proposed algorithms.

We have suggested strong connections to other notorious graph problems such as the Traveling Salesman Problem in its general formulation, as already noted by other authors. While a direct relation among the two problems has not been established yet, we hope we have shed new light on this aspect of structure learning. As the TSP is one of the most studied problems, a reduction among the two problems is of great theoretical and practical importance. In that case, fast techniques for the TSP could be applied also to structure learning, helping in a more efficient network discovery. Furthermore, as almost all of the mathematical programming tools can be applied on the TSP, not to mention its vast pool of cuts, a direct connection may bring a deeper understanding of the problem, and a broader portfolio of techniques to apply.

Finally, as the structure learning of Bayesian Networks is far from being a closed problem, there is also room for new, alternative solutions, either based on IP formulations or on any other approach.

Bibliography

[1] T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.

[2] Hirotugu Akaike. Information theory and an extension of the maximum likelihood principle. In *Second international symposium on information theory*, pages 267–281. Akademinai Kiado, 1973.

[3] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study*. Princeton University Press, 2011.

[4] Silvana Badaloni, Francesco Sambo, and Francesco Venco. Bayesian network structure learning: hybridizing complete search with independence tests. *AI Communications*, 2014.

[5] Mark Bartlett and James Cussens. Advances in bayesian network learning using integer programming. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI 2013)*, pages 182–191, 2013.

[6] Ingo A Beinlich, Henri Jacques Suermondt, R Martin Chavez, and Gregory F Cooper. *The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks*. Springer, 1989.

[7] Richard Bellman. Dynamic programming. 1957.

[8] Timo Berthold, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Thorsten Koch, and Yuji Shinano. Solving mixed

integer linear and nonlinear problems using the SCIP Optimization Suite. ZIB-Report 12-17, Zuse Institute Berlin, Takustr. 7, 14195 Berlin, 2012.

[9] John Binder, Daphne Koller, Stuart Russell, and Keiji Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29(2-3):213–244, 1997.

[10] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[11] E Robert Bixby, Mary Fenelon, Zonghao Gu, Ed Rothberg, and Roland Wunderling. Mip: Theory and practice—closing the gap. In *System modelling and optimization*, pages 19–49. Springer, 2000.

[12] Béla Bollobás. *Modern graph theory*, volume 184. Springer, 1998.

[13] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[14] Nader H Bshouty and Lynn Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In *STACS 98*, pages 298–308. Springer, 1998.

[15] Wray Buntine. Theory refinement on bayesian networks. *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, 91:52–60, 1991.

[16] Alexandra M Carvalho. Scoring functions for learning bayesian networks. 2009.

[17] David Maxwell Chickering. Learning bayesian networks is np-complete. In *Learning from data*, pages 121–130. Springer, 1996.

[18] David Maxwell Chickering. Learning equivalence classes of bayesian-network structures. *The Journal of Machine Learning Research*, 2:445–498, 2002.

[19] David Maxwell Chickering. Optimal structure identification with greedy search. *The Journal of Machine Learning Research*, 3:507–554, 2003.

[20] David Maxwell Chickering and Christopher Meek. Finding optimal bayesian networks. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, UAI'02, pages 94–102, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1-55860-897-4. URL http://dl.acm.org/citation.cfm?id=2073876.2073888.

[21] David Maxwell Chickering, David Heckerman, and Christopher Meek. Large-sample learning of bayesian networks is np-hard. *The Journal of Machine Learning Research*, 5:1287–1330, 2004.

[22] Vašek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete mathematics*, 4(4):305–337, 1973.

[23] Cristina Conati, Abigail S Gertner, Kurt VanLehn, and Marek J Druzdzel. On-line student modeling for coached problem solving using bayesian networks. *Courses And Lectures-International Centre For Mechanical Sciences*, pages 231–242, 1997.

[24] Gregory F Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

[25] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[26] James Cussens. Maximum likelihood pedigree reconstruction using integer programming. In *Proceedings of the Workshop on Constraint Based Methods for Bioinformatics (WCB-10)*, 2010.

[27] James Cussens. Bayesian network learning with cutting planes. *arXiv preprint arXiv:1202.3713*, 2012.

[28] James Cussens. Bayesian network learning by compiling to weighted max-sat. *arXiv preprint arXiv:1206.3244*, 2012.

[29] James Cussens, Mark Bartlett, Elinor M Jones, and Nuala A Sheehan. Maximum likelihood pedigree reconstruction using integer linear programming. *Genetic Epidemiology*, 37(1):69–83, 2013.

[30] Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

[31] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *New York*, 1951.

[32] Sanjoy Dasgupta. Learning polytrees. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 134–141. Morgan Kaufmann Publishers Inc., 1999.

[33] Cassio P De Campos, Zhi Zeng, and Qiang Ji. Structure learning of bayesian networks using constraints. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 113–120. ACM, 2009.

[34] Cassio Polpo de Campos and Qiang Ji. Efficient structure learning of bayesian networks using constraints. *Journal of Machine Learning Research*, 12(3):663–689, 2011.

[35] Luis M De Campos. A scoring function for learning bayesian networks based on mutual information and conditional independence tests. *The Journal of Machine Learning Research*, 7: 2149–2187, 2006.

[36] Luis M de Campos and Juan F Huete. Approximating causal orderings for bayesian networks using genetic algorithms and simulated annealing. In *Proceedings of the Eighth IPMU Conference*, volume 1, pages 333–340, 2000.

[37] Luis M De Campos, Juan M Fernandez-Luna, José A Gámez, and José M Puerta. Ant colony optimization for learning bayesian networks. *International Journal of Approximate Reasoning*, 31(3):291–311, 2002.

[38] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, 1989.

[39] Marco Dorigo and Gianni Di Caro. Ant colony optimization: a new meta-heuristic. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2. IEEE, 1999.

[40] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1-3):23–47, 2003.

[41] Matteo Fischetti and Andrea Lodi. Heuristics in mixed integer programming. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[42] Matteo Fischetti and Michele Monaci. Proximity search for 0-1 mixed-integer convex programming. Technical report, Technical Report, DEI, University of Padova (in preparation), 2012.

[43] Matteo Fischetti, Juan José Salazar González, and Paolo Toth. The symmetric generalized traveling salesman polytope. *Networks*, 26(2):113–123, 1995.

[44] Matteo Fischetti, Andrea Lodi, and Domenico Salvagnin. Just mip it! In *Matheuristics*, pages 39–70. Springer, 2010.

[45] Nir Friedman, Iftach Nachman, and Dana Peér. Learning bayesian network structure from massive datasets: the «sparse candidate «algorithm. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 206–215. Morgan Kaufmann Publishers Inc., 1999.

[46] Maxime Gasse, Alex Aussem, and Haytham Elghazel. An experimental comparison of hybrid algorithms for bayesian network structure learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 58–73. Springer, 2012.

[47] Gabriel Gelius-Dietrich. *cplexAPI: R Interface to C API of IBM ILOG CPLEX*, 2013. URL `http://CRAN.R-project.org/package=cplexAPI`. R package version 1.2.9.

[48] Michel Gendreau and Jean-Yves Potvin. Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140 (1):189–213, 2005.

[49] Krasimira Genova and Vassil Guliashki. Linear integer programming methods and approaches–a survey. *Cybernetics And Information Technologies*, 11(1), 2011.

[50] Fred Glover, Manuel Laguna, et al. *Tabu search*, volume 22. Springer, 1997.

[51] Fred Glover, Manuel Laguna, and Rafael Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 39(3):653–684, 2000.

[52] Ralph E Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[53] David Heckerman, Dan Geiger, and David M Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3):197–243, 1995.

[54] Raymond Hemmecke, Silvia Lindner, and Milan Studený. Characteristic imsets for learning bayesian network structure. *International Journal of Approximate Reasoning*, 53(9):1336–1349, 2012.

[55] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex Analysis and Minimization Algorithms: Part 1: Fundamentals*, volume 1. Springer, 1996.

[56] Saman Hong. *A linear programming approach for the traveling salesman problem*. PhD thesis, Johns Hopkins University, 1972.

[57] Tommi Jaakkola, David Sontag, Amir Globerson, and Marina Meila. Learning Bayesian network structure using LP relaxations. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AI-STATS)*, volume 9, pages 358–365. JMLR: W&CP, 2010.

[58] Donald B Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[59] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[60] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[61] Jessica Kasza and Patty Solomon. A comparison of score-based methods for estimating bayesian networks using the kullback-leibler divergence. *arXiv preprint arXiv:1009.1463*, 2010.

[62] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, 1995.

[63] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[64] Scott Kirkpatrick, D. Gelatt Jr., and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[65] Mikko Koivisto. Advances in exact bayesian structure discovery in bayesian networks. *arXiv preprint arXiv:1206.6828*, 2012.

[66] Mikko Koivisto and Kismat Sood. Exact bayesian structure discovery in bayesian networks. *The Journal of Machine Learning Research*, 5:549–573, 2004.

[67] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models*. MIT Press, 2009.

[68] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[69] Gilbert Laporte and Yves Nobert. Generalized traveling salesman problem through n-sets of nodes-an integer programming approach. *Infor*, 21(1):61–75, 1983.

[70] Pedro Larranaga, Cindy MH Kuijpers, Roberto H Murga, and Yosu Yurramendi. Learning bayesian network structures by searching for the best ordering with genetic algorithms. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 26(4):487–493, 1996.

[71] Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988.

[72] Charles E Leiserson, Ronald L Rivest, Clifford Stein, and Thomas H Cormen. *Introduction to algorithms*. The MIT press, 2001.

[73] Silvia Lindner. *Discrete Optimisation in Machine Learning: Learning of Bayesian Network Structures and Conditional Independence Implication*. PhD thesis, München, Technische Universität München, Diss., 2012, 2012.

[74] John DC Little, Katta G Murty, Dura W Sweeney, and Caroline Karel. An algorithm for the traveling salesman problem. *Operations research*, 11(6):972–989, 1963.

[75] Zhifa Liu, Brandon Malone, and Changhe Yuan. Empirical evaluation of scoring functions for bayesian network model selection. *BMC bioinformatics*, 13(Suppl 15):S14, 2012.

[76] Andrea Lodi. The heuristic (dark) side of mip solvers. In *Hybrid Metaheuristics*, pages 273–284. Springer, 2013.

[77] Brandon Malone, Changhe Yuan, Eric A Hansen, and Susan Bridges. Improving the scalability of optimal bayesian network learning with external-memory frontier breadth-first branch and bound search. *arXiv preprint arXiv:1202.3744*, 2012.

[78] Vittorio Maniezzo, Thomas Stützle, and Stefan Voß. *Matheuristics: hybridizing metaheuristics and mathematical programming*, volume 10. Springer, 2009.

[79] Christopher Meek. Causal inference and causal explanation with background knowledge. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 403–410. Morgan Kaufmann Publishers Inc., 1995.

[80] Christopher Meek. *Graphical Models: Selecting causal and statistical models*. PhD thesis, PhD thesis, Carnegie Mellon University, 1997.

[81] George L Nemhauser and Laurence A Wolsey. *Integer and combinatorial optimization*, volume 18. Wiley New York, 1988.

[82] Teppo Niinimaki and Pekka Parviainen. Local structure discovery in bayesian networks. *arXiv preprint arXiv:1210.4888*, 2012.

[83] Charles E Noon and James C Bean. An efficient transformation of the generalized traveling salesman problem. *Ann Arbor*, 1001:48109–2117, 1989.

[84] Agnieszka Onisko. *Probabilistic causal models in medicine: Application to diagnosis of liver disorders*. PhD thesis, Ph. D. dissertation, Institute of Biocybernetics and Biomedical Engineering, Polish Academy of Science, Warsaw, 2003.

[85] Sascha Ott, Seiya Imoto, and Satoru Miyano. Finding optimal models for small gene networks. In *Pacific symposium on biocomputing*, volume 9, pages 557–567, 2004.

[86] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[87] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[88] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL `http://www.R-project.org/`.

[89] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[90] Sheldon M Ross. *Introduction to probability models*. Access Online via Elsevier, 2006.

[91] Francesca Rossi, Charles J Petrie, and Vasant Dhar. On the equivalence of constraint satisfaction problems. In *ECAI*, volume 90, pages 550–556, 1990.

[92] Edward Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

[93] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 74. Prentice hall Englewood Cliffs, 1995.

[94] Tuhin Sahai, Stefan Klus, and Michael Dellnitz. A traveling salesman learns bayesian networks. *arXiv preprint arXiv:1211.4888*, 2012.

[95] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[96] Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[97] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[98] Tomi Silander and Petri Myllymäki. A simple approach for finding the globally optimal bayesian network structure. *arXiv preprint arXiv:1206.6875*, 2012.

[99] Tomi Silander, Teemu Roos, Petri Kontkanen, and Petri Myllymäki. Factorized normalized maximum likelihood criterion for learning bayesian network structures. 2008.

[100] Ajit P Singh and Andrew W Moore. Finding optimal bayesian networks by dynamic programming. *Technical report, CMU*, 2005.

[101] Stephen Frederick Smith. A learning system based on genetic adaptive algorithms. 1980.

[102] David Sontag. *Approximate Inference in Graphical Models using LP Relaxations*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2010.

[103] David Sontag and Tommi S Jaakkola. New outer bounds on the marginal polytope. In *Advances in Neural Information Processing Systems*, pages 1393–1400, 2007.

[104] David Sontag, Talya Meltzer, Amir Globerson, Yair Weiss, and Tommi Jaakkola. Tightening LP relaxations for MAP using message-passing. In *24th Conference in Uncertainty in Artificial Intelligence*, pages 503–510. AUAI Press, 2008.

[105] DAVID J Spiegelhalter and ROBERT G Cowell. Learning in probabilistic expert systems. *Bayesian statistics*, 4:447–465, 1992.

[106] Daniel Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 296–305. ACM, 2001.

[107] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, prediction, and search*, volume 81. The MIT Press, 2000.

[108] SS Srivastava, Santosh Kumar, RC Garg, and PRASENJIT Sen. Generalized traveling salesman problem through n sets of nodes. *CORS journal*, 7:97–101, 1969.

[109] Milan Studený. *On Probabilistic Conditional Independence Structures*. Springer, 2005.

[110] Milan Studený. Lp relaxations and pruning for characteristic imsets. *Preprint*, 2012.

[111] Milan Studený. Integer linear programming approach to learning bayesian network structure: towards the essential graph. *International Journal of Approximate Reasoning*, 2013.

[112] Milan Studený and David Haws. Learning bayesian network structure: Towards the essential graph by integer linear programming tools. *International Journal of Approximate Reasoning*, 2013.

[113] Milan Studený and Jiří Vomlel. A reconstruction algorithm for the essential graph. *International Journal of Approximate Reasoning*, 50(2):385–413, 2009.

[114] Milan Studený and Jiří Vomlel. On open questions in the geometric approach to structural learning bayesian nets. *International Journal of Approximate Reasoning*, 52(5):627–640, 2011.

[115] Milan Studený, Raymond Hemmecke, and Silvia Lindner. Characteristic imset: a simple algebraic representative of a bayesian network structure. In *Proceedings of the 5th European workshop on probabilistic graphical models*, pages 257–264. Citeseer, 2010.

[116] Milan Studený, Jiří Vomlel, and Raymond Hemmecke. A geometric view on learning bayesian network structures. *International Journal of Approximate Reasoning*, 51(5):573–586, 2010.

[117] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[118] Jin Tian. A branch-and-bound algorithm for mdl learning bayesian networks. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 580–588. Morgan Kaufmann Publishers Inc., 2000.

[119] Ioannis Tsamardinos and Giorgos Borboudakis. Permutation testing improves bayesian network learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 322–337. Springer, 2010.

[120] Ioannis Tsamardinos, Laura E Brown, and Constantin F Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine learning*, 65(1):31–78, 2006.

[121] Francesco Venco. Structural learning of bayesian networks using statistical constraints. Master's thesis, Università degli Studi di Padova, 2012.

[122] Douglas Brent West. *Introduction to graph theory*, volume 2. Prentice hall Englewood Cliffs, 2001.

[123] Shulin Yang and Kuo-Chu Chang. Comparison of score metrics for bayesian network learning. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 32(3): 419–428, 2002.

Acknowledgements

I thank prof. Silvana Badaloni for being my advisor for this thesis. I'm of course indebted with dr. Francesco Sambo, who guided me while letting me freedom of tinkering. I've had fun, and learnt a lot.

I need also to thank my coursemates, for the time spent (and the help with the projects), and my family and friends for support.

Finally, I thank all of those who, inside or outside schools and university, taught me something. I tried to learn from many.