TÉCNICO LISBOA

From cause-effect pairs to causal graphs

Margarida Freitas de Sá Mendes

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Prof. Mário Alexandre Teles de Figueiredo

Examination Committee

Chairperson: Prof. José Eduardo Charters Ribeiro da Cunha Sanguino
Supervisor: Prof. Mário Alexandre Teles de Figueiredo
Member of the Committee: Prof. Susana de Almeida Mendes Vinga Martins

November 2023

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to start by thanking my supervisor, Mário Figueiredo for all the guidance and support throughout the process of writing this thesis, the knowledge shared in the last few months was crucial for the completion of this work. I also want to thank my family for their immeasurable support and for believing in me even at times when I couldn't do it. Finally, I would like to thank my friends for their companionship and motivational coffees shared during these past months.

Resumo

A inferência de causalidade é um dos principais tópicos no mundo da aprendizagem automática, sendo a procura de desvendar as causas e efeitos subjacentes em sistemas complexos um desafio considerável. O objetivo fundamental da inferência de causalidade é desvendar a intrincada teia de relações entre variáveis, revelando os mecanismos e processos que as ligam. Esta procura de causalidade deu origem a um conjunto diversificado de métodos, cada um oferecendo uma perspetiva única para explorar e compreender estas ligações. Ao longo dos anos, foram desenvolvidos vários métodos para tentar resolver o problema da descoberta da causalidade, alguns dos quais podem ser classificados em três grupos principais: métodos baseados em restrições, métodos baseados em pontuações (”scores”) e métodos concebidos para distinguir entre pais e filhos nas relações causais. Os métodos baseados em restrições procuram descobrir ligações causais examinando as relações de independência condicional nos dados. Os métodos baseados em pontuações, por outro lado, empregam medidas estatísticas para avaliar a força das relações causais, e os métodos que distinguem pais e filhos concentram-se em delinear a direção da causalidade entre as variáveis. Esta tese tem como objetivo explorar o panorama da inferência causal, com um foco específico na análise e comparação de diferentes métodos baseados em pontuações, incluindo o Bayesian Information Criterion (BIC) amplamente utilizado e o Bayesian Dirichlet equivalent uniform (BDeu), juntamente com as suas respectivas variantes implementadas, utilizando a entropia e a divergência de Kullback-Leibler. O objetivo é esclarecer o desempenho destes scores no contexto de vários cenários de descoberta causal, em diferentes tipos de grafos caracterizados por estruturas distintas e com diferentes níveis de ruído no sistema. Ao dissecar e analisar estes métodos através de uma série de configurações de grafos, esta tese procura fornecer uma compreensão mais profunda dos pontos fortes e das limitações de várias técnicas de inferência causal, facilitando uma análise causal mais precisa e robusta em aplicações práticas.

Palavras-chave: Inferência de causalidade, métodos baseados em scores, Bayesian Information Criterion (BIC), Bayesian Dirichlet equivalent uniform (BDeu).

Abstract

Causal Inference is one of the current primary topics in machine learning, the pursuit of unraveling the underlying causes and effects in complex systems being a considerable challenge. The fundamental goal of causal inference is to disentangle the intricate web of relationships between variables, revealing the mechanisms and processes that link them together. This quest for causality has given rise to a diverse array of methods, each offering a unique lens through which to explore and understand these intricate connections. Several methods have been developed over the years to try and solve the causality discovering issue, some of these can broadly be categorized into three main groups: constraint-based methods, score-based methods, and methods designed to distinguish between parents and children in causal relationships. Constraint-based methods seek to unearth causal links by examining conditional independence relationships in the data. Score-based methods, on the other hand, employ statistical scores to assess the strength of causal relationships, and methods distinguishing parents from children focus on delineating the direction of causality between variables. This thesis embarks on a comprehensive exploration of the causal inference landscape, with a specific focus on analyzing and comparing different scoring methods, including the widely employed Bayesian Information Criterion (BIC) and the Bayesian Dirichlet equivalent uniform (BDeu) score, along with their respective implemented variants, using entropy and Kullback-Leibler divergence. The objective is to shed light on how these scores perform in the context of various causal discovery scenarios, as we examine different types of graphs characterized by distinct structures and introduce varying levels of noise into the system. By dissecting and scrutinizing these methods through an array of graph configurations, this thesis seeks to provide a deeper understanding of the strengths and limitations of various causal inference techniques, facilitating more precise and robust causal analysis in practical applications.

Keywords: Causal inference, Score-based methods, Bayesian Information Criterion (BIC), Bayesian Dirichlet equivalent uniform (BDeu).

Contents

List of Tables

List of Figures

Acronyms

ANM Addictive Noise Model

BAMB Balenced MB

BD Bayesian Dirichlet

BDeu Bayesian Dirichlet with score equivalence and uniform priors

BIC Bayesian Information Criterion

BN Bayesian Network

CMB Candidate Markov Blanket

DAG Directed Acyclic Graph

EEMB Efficient and Effective MB

FBEDk Forward-Backward selection with Early Dropping

FCM Functional Causal Models

GSMB Grow-Shrink Markov Blanket

IAMB Incremental Association Markov Blanket

IFBS Intervaling Forward-Backward Selection

KL Kullback-Leibler

MB Markov Blanket

MMMB Min-Max MB

MMPC Max-Min Parents and Children

PC Parent Children

PCMB Parents and Children based MB

PDAG Partially Directed Acyclic Graph

RSS Residual Sum of Squares

SCM Structural Causal Model

SE Structural Equation

SEM Structural Equation Model

SFBS Standard Forward-Backward Selection

SLL Score-based Local Learning

SP Spouses

STMB Simultaneos MB

TIE* Target Information Equivalence

TSS Total Sum of Squares

Chapter 1

Introduction

1.1 Causality

Causality is a fundamental concept at the heart of understanding the relationships between events and phenomena, has been a subject of profound fascination and debate throughout the history of human thought, the pursuit of unraveling causal connections has shaped our comprehension of the world.

Causal relations are of extreme importance to a number of different scientific fields since they enable us to predict what the consequence of a certain action will be [1]. Its relevance is undeniable, as causal relationships are pivotal for making informed decisions, effecting change, and predicting future outcomes. Because of its importance, in the past years, causality research experienced a major evolution, converting from a concept into a well-defined mathematical object [1], and methods to infer causal relationships have been in continuous development.

At its core, causal discovery seeks to transcend mere correlation and dive into the realm of causation. It grapples with the fundamental challenge of inferring causality from observed data, a task that has captivated scholars and researchers for centuries. With the advent of advanced computational techniques and the explosion of big data, the field has witnessed remarkable growth and transformation.

Fundamentally, the problem of causal discovery can be distilled into several key issues, each of which presents unique hurdles to researchers, scientists, and data analysts. Understanding these challenges is essential to appreciate the scope of the problem.

The first relevant issue is data limitation: causal discovery relies on observational data, which inherently reflects the correlations among variables but not the underlying causal relationships. Limited data can result in inaccurate or incomplete causal inferences. Obtaining comprehensive data with the ideal experimental setup is often impractical, especially when studying complex real-world systems. Besides this, the presence of confounding variables is also something to be attentive to since confounders are variables that are correlated with both the cause and the effect, creating spurious correlations. Disentangling the true causal link from these confounding variables is a fundamental challenge in causal discovery, since ignoring confounders can lead to erroneous causal inferences. Apart from these two main issues, indirect effects, feedback loops, and nonlinear relationships should also not be ignored since the

first complexities can make it challenging to establish direct causal links between variables, especially when considering dynamic systems or cyclic dependencies, and the second may lead to issues in capturing the intricate cause-and-effect dynamics present in various systems, using linear models. Finally, the complexity and scalability of the data may also impose a challenge, since for large-scale systems, the number of potential causal relationships can be vast. Causal discovery algorithms must be both efficient and scalable to handle the dimensionality and complexity of modern data.

In the quest to understand causality, statistical methods have proven very useful for uncovering connections between variables. However, they face a significant challenge - they can't fully reveal causality on their own. This challenge arises because causation is a complex and multifaceted concept that goes beyond mere statistical associations.

Statistics can tell us when variables are linked or correlated, but they can't tell us the underlying cause-and-effect mechanisms that create these links. Causality involves understanding the deeper processes and reasons behind these connections, and this goes beyond what statistics can provide.

Because of this, methods to infer these causal relations have been developed in the past decades. These methods aim to achieve the goal of causal discovery: given observed data, infer as much as possible about the generating mechanism, particularly, (since we'll describe the causal problem as a graph) infer the causal graph [8].

1.2 Related Concepts

In order to further discuss the topic of causality, causal inference, and causal models, it's important to have some specific concepts in mind. These will be discussed in detail later on so that full background on the topic is given. However, a few introductory ideas will be mentioned here, namely: Bayesian Networks, Directly Acyclic Graphs, Markov Blankets, and the Common Cause Principle

1.2.1 Directed Acyclic Graphs

DAGs are graphical structures that provide a powerful way to represent relationships and dependencies among a set of random variables. Each node in a DAG represents a variable, and directed edges between nodes indicate the presence of direct causal or probabilistic relationships. The directed nature of the edges implies that they have a specific direction, from one node to another, showing the influence or dependence of the receiving variable on the sending variable [2].

The "acyclic" part of Directed Acyclic Graphs is fundamental. It means that no closed loops or cycles exist within the graph. In other words, you can start at any node and follow the directed edges, but you can never return to the same node by following those edges. This acyclic property ensures that there are no feedback loops or self-referential relationships, making DAGs an ideal tool for representing acyclic cause-and-effect structures [6].

DAGs are particularly valuable for causal inference and probabilistic modeling. They provide a clear and interpretable way to express cause-and-effect relationships. When dealing with complex systems

or datasets, DAGs allow researchers and data scientists to visualize and understand the underlying mechanisms at play.

1.2.2 Bayesian Networks

Bayesian networks (BNs) are powerful graphical models that provide a structured way to represent and analyze complex probabilistic relationships. These networks are widely used in various fields, including machine learning, artificial intelligence, and statistics, to tackle problems involving uncertainty and causal inference.

At the heart of a BN is a DAG, as discussed earlier. However, unlike causal DAGs, BNs are not always intended to represent causal relationships. Instead, they capture conditional dependencies between variables. This means that the existence of an edge between two nodes represents that one variable depends on the other in a probabilistic sense [31].

The key strength of BNs lies in their ability to model complex systems while offering a structured, interpretable representation. They excel in situations where variables interact with one another, and the relationships between these variables can be probabilistic and uncertain. Using Bayesian Networks, we can answer questions like ”given this information, what’s the probability of that event occurring?”

Each node in a BN contains a conditional probability table or function, specifying how the variable represented by that node depends on its parent nodes. These conditional probabilities capture the probabilistic relationships among the variables in the network. BNs can be used for various tasks, including probabilistic inference, reasoning under uncertainty, classification, and decision support.

1.2.3 Markov Blanket

The Markov Blanket is a fundamental concept in the realm of BNs and probabilistic graphical models. It plays a crucial role in understanding the conditional independence relationships between variables within a network.

In essence, the Markov Blanket of a particular node in a BN is the minimal set of nodes that, when observed, renders that node conditionally independent of all the other nodes in the network. This concept of conditional independence is vital for probabilistic modeling because it allows us to focus on the most relevant variables without considering the entire network, making computations more tractable [2].

The Markov Blanket can be thought of as a protective shield for a given node within the network. If you know the values of the variables in the Markov Blanket, you effectively ”block” all probabilistic influences from the rest of the network, ensuring that the node of interest is conditionally independent of other nodes.

In a Bayesian Network, the Markov Blanket of a node typically consists of three types of nodes [2]:

- **Parents:**These are the immediate predecessors of the node in question. Knowing the parents helps you understand how the node depends on its direct causes.

- **Children:** These are the nodes directly influenced by the node in question. Understanding the children is crucial for assessing the consequences of interventions on the node of interest.

- **Parents of Children:** These are the parents of all the children of the node. These variables are necessary because they shield the node from influences that may come through its children.

The concept of a Markov Blanket has a wide range of applications, from simplifying probabilistic inference in Bayesian Networks to causal discovery and feature selection in machine learning. Identifying the Markov Blanket of a variable is a powerful tool for focusing on the most relevant information, which is particularly important when dealing with large and complex datasets.

1.2.4 Reichenbach's common cause principle

Reichenbach's Common Cause Principle, developed by the influential philosopher Hans Reichenbach (1891-1953), stands as a foundational concept in the realm of causality. It tackles the intriguing interplay between cause and effect, offering insights into the relationship between correlated events.

The essence of this principle lies in the idea that when two events, denoted as A and B, exhibit a correlation, it implies the existence of a common underlying cause or factor that influences both events. In simpler terms, it challenges the notion that all correlations should be interpreted as direct causations. Instead, it suggests that these correlations can often be elucidated by the presence of an unobserved common cause that impacts both events simultaneously [2].

Reichenbach's Common Cause Principle serves as a valuable instrument for distinguishing genuine causation from mere correlation.

1.3 Objectives and Deliverables

The first goal of the thesis is to conduct a review of causality, laying a solid foundation for the subsequent research. It delves into the statistical, and practical aspects of causation, unraveling the intricate nature of cause-and-effect relationships.

Building upon this foundation, the thesis transitions into its second goal, which is to review various methods of causal inference. It explores different types of techniques that have been developed to tackle the complex task of deciphering causal relationships from data.

The third goal of the thesis takes the research a step further by proposing the implementation of new scores in addition to the existing ones, the final goal being evaluating different scores used in causal discovery. This goal involves the practical application of diverse scoring methods to different datasets, assessing their performance under varying conditions. By subjecting these scores to rigorous evaluation, the thesis provides a comparative analysis that informs about the strengths and limitations of each approach.

1.4 Thesis Outline

In Chapter 2, we start by reviewing some basic concepts about causality the problem of inferring causal relations, and a few basic concepts. From Sections 2.1 to 2.4 the concepts of Directed Acyclic Graphs, Markov Blankets, Additive noise Models, and the overall cause-effect problem are presented in order for a base of knowledge to be laid out for the work developed in this thesis. From Sections 2.5 to 2.7 different methods of inferring causality are presented and studied, so that one can be chosen to test the implementations worked further on. In Chapter 3, the implementation of the project itself is described. In Section 3.1, the choice of the toolbox is described; in Section 3.2, the functions already implemented in the chosen toolbox are presented in order to show the laying ground for changes made in the following sections. In Section 3.3, the new implemented scores are described with each respective pseudocode presented. The results for the tests done using these new implemented scores are presented in Chapter 4. Finally, in Chapter 5 some final conclusions are drawn and remarks done, some future work ideas are proposed.

Chapter 2

Background

2.1 Direct Acyclic Graphs

We start by considering a finite family of random variables, **X**= $\{X_1, ..., X_p\}$, indexed using the set **V**= $\{1, ..., p\}$. The joint distribution is denoted as $\mathscr{L}$(**X**). A graph $\mathcal{G}(\mathbf{V}, \mathcal{E})$ consists of nodes $\mathbf{V}$ and edges $\mathcal{E} \subseteq \mathbf{V}^2$ with $(\nu, \nu) \notin \mathcal{E}$ for any $\nu \in \mathbf{V}$. A node *i* is a parent of *j* if (i,j) $\in \mathcal{E}$ and a child if (j,i) $\in \mathcal{E}$. The set of parents of j is denoted $PA_j^{\mathcal{G}}$ and the set of children $CH_j^{\mathcal{G}}$ [2].

A *partially directed acyclic graph* **(PDAG)** is a graph with no directed cycles, meaning that there is no pair (j,k) such that there are directed paths from j to k and from k to j [6]. After understanding this, it is easy to understand the concept of a *directed acyclic graph* **(DAG)**, which is no more than a PDAG in which all the edges are directed, as observed in figure 2.1.

In a causal DAG $\mathcal{G}$ each variable x_i is associated with a node *i*.

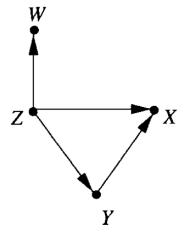


Figure 2.1: Example of a DAG [1]

2.1.1 Structural Equation Models

A structural equation model (SEM), also known as a function model, is a tuple $(\mathcal{S}, \mathcal{L}(N))$ in which $\mathcal{S} = (S_1, ...S_p)$ is a set of *p* equations,

$$S_j : X_j = f_j(\mathbf{PA}_j, N_j), j = 1, ...p, \tag{2.1}$$

and $\mathcal{L}(N) = \mathcal{L}(N_1, ..., N_p)$ is the joint distribution of the noise variables. We can obtain the graph of a structural equation model by drawing direct edges from each parent to its direct effects. This means that

a SEM gives us the information on how $\mathbf{PA}_j$ affects X_j [6].

Structural Causal Models In the case of each variable having a distinct equation in which it appears on the dependent variable side, the model is called a Structural Causal Model (SCM) [1]. These are also known as non-parametric SEM and are usually used for causal modeling purposes [26].

Noise Variables The noise variables mentioned previously, are formally defined in a way that implies that N_j exclusively influences the target variable X_j, without directly affecting any other variables. They represent observed or unobserved background factors that are kept unexplained. Furthermore, since N_j is unobserved it symbolizes a "phenomenon" that prevents X_j from being entirely explained in a deterministic manner by the observed values of its parent variables [27].

2.2 Markov Blanket

A relevant concept when trying to predict the value of a variable based on observed values of all the other values is the Markov Blanket: it ensures that when knowing the Markov Blanket of a variable Y, the other variables do not provide any further information on Y [2]. It can be formally defined as:

Definition 2.2.1 (Markov Blanket)**.** Consider a DAG $\mathcal{G}(\mathbf{V}, \mathcal{E})$ and a target node **Y**. The Markov Blanket of **Y** is the smallest set of variables M such that $\mathbf{Y} \perp\!\!\!\perp \mathbf{V} \backslash (\{\mathbf{Y}\} \cup M)$ given M. If $\mathbf{P}_x$ is Markovian with respect to $\mathcal{G}$, then $\mathbf{Y} \perp\!\!\!\perp \mathbf{V} \backslash (\{\mathbf{Y}\} \cup M)$ given M.

This definition can be explained as follows: the Markov Blanket of a variable **Y** contains its parents, children, and the other parents of its children, also named its spouses. It is easy to understand when visualized in an example as the one in Figure 2.2.

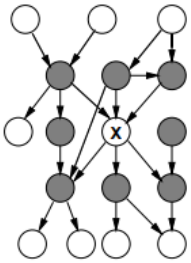


Figure 2.2: Example of a Markov Blanket [10].

After understanding the concept of the Markov Blanket, it can be used to justify Reichenbach's common cause principle:

Proposition 1 (**Reichenbach's common cause principle**)**.** *Assume that any pair of variables X and Y can be embedded into a larger system in the following sense. There exists a correct SCM over the collection* ***X*** *of random variables that contain X and Y with graph* $\mathcal{G}$*. Then Reichenbach's common cause principle follows from the Markov property. If X and Y are (unconditionally) dependent, then there is*

- *either a direct path from X to Y, or*
- *from Y to X, or*
- *there is a node Z with a directed path from Z to X and from Z to Y [2].*

2.2.1 Causal Relationships and Interventions

A causal relationship is a connection or association between two or more variables or events where one (the cause) is responsible for producing a change, effect, or outcome in another (the effect). In a causal relationship, a change in the cause leads to a predictable change in the effect, implying a cause-and-effect connection [28].

The difficulty with the discovery of causal relationship is that the default approach (and the only that does not require further assumptions) is to adopt an experimental method and this is an option that in many cases may not be available for being expensive, unethical, or maybe just impossible. Because of this, interventions are used to simulate physical interventions using a mathematical operator *do(x)*, by deleting certain functions from the model and replacing them with a constant **X**=*x* while the rest of the model remains unchanged. The post-intervention distribution resulting from the action $do(X = x)$ is given by [9].

$$P_M(y|do(x)) = P_{Mx}(y). \tag{2.2}$$

In order to mathematically express the causal relations between pairs of variables, we can write structural equations (SEs) also named functional causal models (FCMs), which help formalize causal relations instead of representing a model that only reproduces the observed joint probability distributions [3]. This is represented by when having a causal relation in which X causes Y $(X \to Y)$, having a SE called an 'assignment' [2],

$$Y := f_Y(X, N_Y), \text{ with } N_Y \perp\!\!\!\perp X, \tag{2.3}$$

in which N_Y is an unobserved noise term. As mentioned before, the fact that equation (2.3) is a SE means that a causal relation is formalized, which implies that a random variable Y is obtained if X is set to x by an external intervention given by

$$Y^{do(X=x)} = f(x, N_Y). \tag{2.4}$$

With this, interventional probabilities can be computed using (2.4) when the causal direction is known [3]. In spite of the way that the cause-effect problem is usually presented, not having in account the task of inferring the structural equation of the form of equation 2.3, various approaches to address this problem have been based on fitting structural equations. An example of this is additive-noise-based causal inference, which will be discussed further but that, briefly, amounts to fitting two different regressions and deciding the causal direction based on which of the regressions residuals is more independent of

its corresponding cause variable [3].

2.3 The Cause-effect Problem

The cause-effect problem can be simply be stated as that of deciding, having two dependent variables X and Y, whenever X causes Y or vice versa [3]. The goal is to discover what is the causal relationship between the two variables, which will enable the prediction of the effects of actions that perturb the observed system [4]. We describe the problem as follows:

> Given observations $(x1,\ y_1)$, . . . , $(x_k,\ y_k\ )$ iid drawn from some distribution $P_{X,Y}$, infer whether X causes Y or Y causes X, given the assumption that exactly one of these alternatives is true [3].

We can assume, in this context, that the observed statistical dependencies between X and Y (the relationship between the two variables that indicates how changes in one variable may be associated with changes in another) are due to the influence of one of the said variables on the other and not due to impact of a third variable, meaning that there is no significant confounding [3]. This means, that in a situation where an extraneous variable, also known as a confounder, is associated with both the independent variable (or predictor) and the dependent variable (or outcome) under investigation, this extraneous variable can lead to a false or misleading association between the independent and dependent variables. Confounding can significantly impact the validity of research findings and the ability to establish causal relationships [29].

2.4 Additive Noise Models

As mentioned before, we aim to learn a causal DAG from an observational joint distribution. Most previous methods have used linear approaches for the case where the observed data is continuous, not necessarily because the causal relation is linear, but because linear models are well understood, very well developed, and easy to work with. For continuous variables, a linear model with additive Gaussian noise is assumed; however the non-Gaussianity in the data can sometimes help discovering the direction of the edges in the DAG, that is, to distinguish the causal directions [8]. That said, if the observational distribution follows a SEM with an additive noise structure the DAG becomes identifiable [6].

Identifiability assures that, after obtaining an infinite number of observations from a model, it is theoretically possible to learn the true values of its underlying parameters. This is a property that must be satisfied in order for inference to be achievable. Identifiability is important since if several models can generate the same distribution, the true model may not be identifiable, even in the ideal case of having access to an infinite amount of data [1]. Formally, identifiability is defined as follows:

Definition 2.4.1 (Identifiability)**.** Let $Q(M)$ be any computable quantity of model M. We say Q is identifiable in a class $\mathbf{M}$ of models if, for any pairs of models M_1 and M_2 from $\mathbf{M}$, $Q(M_1) = Q(M_2)$ whenever

$P_{M_1}(v) = P_{M_2}(v)$. If our observations are limited and permit only a partial set F_M of features (of $P_M(v)$) to be estimated, we define Q to be identifiable from F_M if $Q(M_1) = Q(M_2)$ whenever $F_{M_1} = F_{M_2}$ [1].

Classic methods usually assume faithfulness, identifying only the Markov equivalence class of the graph, which means that some edges will be left undirected. Faithfulness is defined as:

Definition 2.4.2 (Faithfulness)**.** The graph $\mathcal{G}$ of some BN *N* is faithful to a joint probability distribution *J* over feature set V if and only if every dependence entailed by $\mathcal{G}$ is also present in *J*. We say that a data-generating process K is faithfully represented by *N*, if K in the sample limit produces data with joint probability distribution P, and *N* is faithful to P. A BN *N* is faithful if there is a probability distribution *J* to which it is faithful [11].

Another important concept to have in mind is that of the modularity assumption also known as Independence of Causal Mechanism (ICM), it is what allow us to represent different interventional distributions into one graph. It can be formally defined as:

Definition 2.4.3. If we intervene on a set of nodes $S \subseteq [n]^1$, setting them to constants, then for all i, we have the following;

1. If $i \notin S$, then $P(x_i|\mathbf{PA}_\mathbf{i}^\mathcal{G})$ remains unchanged.
2. If $i \in S$, then $P(x_i|\mathbf{PA}_\mathbf{i}^\mathcal{G}) = 1$ if x_i is the value that X_i was set to by the intervention; otherwise, $P(x_i|\mathbf{PA}_\mathbf{i}^\mathcal{G}) = 0$ [18].

Consider a model to test, for example, if there is a causal relationship from X to Y, as

$$Y = f(X) + N, \tag{2.5}$$

in which f is a linear function and N an additive noise (independent of X). In most of the cases $P^{(X,Y)}$ only admits such a model in one of the directions ($X \to Y$ or $Y \to X$), unless we are dealing with a bivariate Gaussian case. Because of how specific the case has to be for this to happen, we usually only have such an ANM in one direction and not on the other, meaning that we can infer the first as the causal direction [5]. We can formalize this as:

ANM Causal Inference: *Whenever Y satisfies an additive noise model with respect to X and not vice versa, then we infer X to be the cause for Y and we write* $X \to Y$ [5].

2.5 Constraint-Based Methods

Constraint-based methods are established having in account two propositions. In the first one, the relations between a node and its parents or children is explained. We conclude that if V_i is a parent or a child of V_j in a Bayesian Network (BN), these two variables are not independent conditioning on any subsets of V $\setminus\{ V_i, V_j\}$ [7].

Proposition 2. *In a BN, if node V_i is a parent (or a child) of V_j, then $\forall S \subseteq V \setminus \{V_i, V_j\}$, $V_i \not\perp\!\!\!\perp V_j | S$*

Secondly, we are presented with the relations between a node and its respective spouses in a Bayesian Network. The proposition means that if V_i is a spouse of V_k and V_j is their child in common, we have a subset $S \subseteq V \setminus \{V_i, V_j, V_k\}$ in which V_i and V_j are independent given S, but dependent given S $\cup V_j$. This means that given a node, its spouse and common child together contain more predictive information about the node than the common child alone. It also adds that the spouses of V_k are all of the parents of the children of V_k [7].

Proposition 3. *In a BN, assuming that V_i is adjacent to V_j, V_j is adjacent to V_k, and V_i is not adjacent to V_k (e.g., $V_i \rightarrow V_j \leftarrow V_k$), if $\exists\ S \subseteq V \setminus \{V_i, V_j, V_k\}$ such that $V_i \perp\!\!\!\perp V_k | S$ and $V_i \not\perp\!\!\!\perp V_k | \{S, V_j\}$ hold, V_i is a spouse of V_k.*

Statistical independence tests are used by current methods as a selection criteria that decides if we should add or remove features from S. Five different types of tests are used as the mentioned criteria in constraint-based methods: λ^2 test, G^2 test, mutual information for discrete features, Fisher's Z test for continuous features with linear relations and additive Gaussian errors, and kernel based tests for continuous features with non-linearity and non-Gaussian noise [7].

2.5.1 Simultaneous MB learning

With this type of algorithm, we're given a class C and the goal is to simultaneously find the parents, children and spouses while learning the Markov Blanket. It greedily learns a MB of class C by conditioning on an entire estimate candidate Markov Boundary of C by assuming a forward-backwards strategy [7]. The general Simultaneous MB learning algorithm pseudocode is presented in figure 2.3,the forward phase corresponds to lines 5 to 9 and the backward phase goes from line 12 to line 16.

ALGORITHM 1: The Instantiation of SFBS for Simultaneous MB Learning

1: **Input**: Feature Set F and the class variable C
   Output: $CMB(C)$
2: $CMB(C) = \emptyset$;
3: //Forward phase: Adding candidate MB (relevant) features to $CMB(C)$
4: **repeat**
5: Select a feature $X \in F$;
6: **if** $X \not\perp\!\!\!\perp C | CMB(C)$ **then**
7: $CMB(C) = CMB(C) \cup X$ and $F = F \setminus X$;
8: **end if**
9: **until** no features in F are added to $CMB(C)$;
10: //Backward phase: Removing false positives from $CMB(C)$
11: **repeat**
12: Select a feature $Y \in CMB(C)$;
13: **if** $Y \perp\!\!\!\perp C | CMB(C) \setminus Y$ **then**
14: $CMB(C) = CMB(C) \setminus Y$;
15: **end if**
16: **until** no features in $CMB(C)$ are removed
17: Output $CMB(C)$

Figure 2.3: Pseudocode of a Simultaneous MB learning algorithm [7]

This section delves into three prominent techniques, each with its distinct approach: the Grow-Shrink Markov Blanket, the Incremental Association Markov Boundary , and the Forward-Backward selection with Early Dropping. These methods are equipped with heuristics which are problem-solving or decision-making strategies that employs practical, often simplified, rules or methods to find a solution or make a judgment quickly and efficiently, even when the solution may not be guaranteed to be optimal or perfect [30].

GSMB **Grow-Shrink Markov Blanket (GSMB)** was the first method that didn't learn the entire Bayesian Network. It is divided, as mentioned before, into two phases: a forward and a backward one. The forward phase corresponds to a **growing phase**, in which while the Markov Blanket property of X is violated, meaning that we have a variable in V that depends on X, this variable will be added to the current set S until we have no such variables left. However, this may add a few variables to S that do not belong to the Markov Blanket, even though these would be classified as independent at a later stage once "intervening" nodes of the underlying Bayesian net are added to S. This is where the **shrinking phase** enters, since it identifies and removes the variables that do not belong to the set. This means that the $\mathcal{O}(n)$ conditional tests runtime is $\mathcal{O}(nD)$, in which $n = |V|$, and D is the set of examples [10]. It may be useful to minimize the number of tests by using a heuristic to order the variables, either ordering them by ascending mutual information, or even by probability of dependence between X and Y (using, for example, the χ^2 test) [10]. However, either one of these examples represents a static heuristic that may add many false positives to $CMB(C)$ in the first phase of the algorithm, growing the size of the estimate of the Markov Blanket. For this reason, GSMB is ineffective when it comes to data sets with small sample sizes but with high dimensionality [7].

IAMB The **Incremental Association Markov Boundary (IAMB)** is an improved version of the GSMB that aims to solve the problem mentioned above, using a dynamic heuristic in the forward phase. At each iteration, the algorithm adds to the estimate of the Markov Blanket the feature $X \in F \setminus \text{CMB}(C)$ with the highest association to the class variable, conditioning on the CMB(C) at the time if $X \perp\!\!\!\perp C|\text{CMB}(C)$ holds. This allows the features that truly belong to the Markov Blanket of C to enter the CMB(C) as early as they can, reducing the chance of false positives entering the estimated Markov Blanket in the forward phase [7].

In order to identify the Markov Blanket of the target variable, the algorithm makes two assumptions:

1. The data is generated by processes that can be faithfully represented by a Bayesian Network;
2. We have a reliable statistical conditional independence tests so it is possible to asses the independence and strength of association of the target variable with some other variable given a set of variables [11].

The way the heuristic previously mentioned works, is by starting with an empty set for the Markov Boundary of the target variable, and in the following iteration, the variable that has the largest association to the target variable conditioned on CMB is admitted to the set. This process is stopped when each

variable F in the CMB and the target variable are independent. Since all variables that belong to the MB will eventually enter the CMB, the heuristic is admissible.

In spite of IAMB having a lower time complexity and data sample requirement when compared to the GSMB, the number of data samples required is still exponential with the size of CMB(C). In order to try and diminish the issue, a few variants of IAMB were proposed, including IAMBnPC, Inter-IAMB, Inter-IAMBnPC and Fast-IAMB. Inter-IAMB and Inter-IAMBnPC use the IFBS framework, contrary to the previous algorithms that used the SFBS framework. The goal of these two methods is to keep the size of the CMB(C) set as small as it is possible. With a different approach, Fast-IAMB tries to make the forward phase more efficient by adopting a greedy strategy, adding as many features conditionally dependent on C, until an independence test is not reliable [7].

FBED^k Forward-Backward selection with Early Droping ($FBED^k$) was also created based on IAMB, where the IAMB would reconsider all remaining features, this algorithm uses a strategy of early dropping during the forward phase, removing, at each iteration, the features that are conditionally independent of C given the current CMB(C). Its pseudocode is presented in 2.4. The purpose is to quickly reduce the number of variables that are candidates and still keep the relevant ones [12]. It terminates once the features set is empty, and the forward phase runs K additional times to reconsider previously dropped features. The backwards phase is then applied identically to IAMB [7].

2.5.2 Divide and conquer MB learning

As previously said, simultaneous MB learning methods have a data requirement issue. In order to solve this, a divide and conquer approached was created. It divides the learning of the Markov Blanket problem into two sub-problems [7]:

- learning of the parents and children (PC(C))
- learning of spouses (SP(C))

In the first sub-problem we do not use the entire PC(C) as the conditioning sets, it rather uses subsets of it, meaning that it needs a much smaller number of samples in comparison to the simultaneous learning method. For example, in order to determine if a feature (X) is a candidate member of the set of parents and children, this approach checks possible subsets of PC(C) and if a subset $S \subseteq PC(C)$ exists such that $X \perp\!\!\!\perp C|S$ holds. This exploring process terminates and X will be discarded and never considered again, contrary to what happens in the simultaneous learning approach in which discarded features will be reconsidered a lot of times. In spite of being data efficient, the divide and conquer approach is not time effcient, it mitigates the bigger sample requirement however, the algorithms that exist for this method will become computationally expensive when the size of currently selected features become large [7].

In this section, we will explore three distinct methods: the Min-Max Markov Blanket (MMMB), the Parents and Children-based Markov Blanket (PCMB), and the Simultaneous Markov Blanket (STMB)

Algorithm 2 Forward-Backward Selection with Early Dropping (FBED^K)

Input: Dataset $\mathcal{D}$, Target T, Maximum Number of Runs K
Output: Selected Variables $\mathbf{S}$

$\mathbf{S} \leftarrow \emptyset$ *//Set of selected variables*
$K_{cur} \leftarrow 0$ *//Initializing current number of runs to 0*

//Forward phase: iterate until (a) run limit reached, or (b) $\mathbf{S}$ *does not change*
while $K_{cur} \leq K \wedge$ $\mathbf{S}$ changes **do**
    $\mathbf{S} \leftarrow$ OneRun$(\mathcal{D}, T, \mathbf{S})$
    $K_{cur} \leftarrow K_{cur} + 1$
end while

//Perform backward selection and return result
return BackwardSelection$(\mathcal{D}, \mathbf{T}, \mathbf{S})$

function OneRun($\mathcal{D}$, T, $\mathbf{S}$)
    $\mathbf{R} \leftarrow \mathbf{V} \setminus \mathbf{S}$ *//Set of remaining candidate variables*

    //Forward phase: iterate until $\mathbf{R}$ *is empty*
    while $|\mathbf{R}| > 0$ **do**
        //Identify best variable V_{best} *out of* $\mathbf{R}$*, according to* Perf
        $V_{best} \leftarrow \underset{V \in \mathbf{R}}{\text{argmax}}\ \text{Perf}(\mathbf{S} \cup V)$
        //Select V_{best} *if it increases performance according to criterion* C
        if $\text{Perf}(\mathbf{S} \cup V_{best}) \underset{\text{C}}{>} \text{Perf}(\mathbf{S})$ **then**
            $\mathbf{S} \leftarrow \mathbf{S} \cup V_{best}$
        end if
        //Drop all variables from $\mathbf{R}$ *not satisfying* C
        $\mathbf{R} \leftarrow \{V : V \in \mathbf{R} \wedge V \neq V_{best} \wedge \text{Perf}(\mathbf{S} \cup V) \underset{\text{C}}{>} \text{Perf}(\mathbf{S})\}$
    end while
    return S
end function

Figure 2.4: Pseudocode of the FBEDk algorithm [12]

algorithm. These methods embrace the Divide and Conquer philosophy, where the process of Markov Blanket discovery is tackled in stages, revealing insights into the conditional dependencies of variables and are grounded in the Standard forward-backward selection (SFBS) framework, that plays a crucial role in their operation, as it guides the identification of relevant Markov Blankets [7].

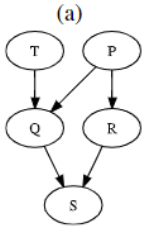
MMMB The Min-Max MB (MMMB) algorithm starts by finding candidates for the parents and children set resorting to the Max-Min Parents and Children (MMPC). This last algorithm uses the SFBS framework and begins in the forward phase where variables enter the set of PC(C) set sequentially by using an heuristic, posteriorly in the backwards phase it removes all false positives [13]. However, because we want to identify the Markov Blanket which includes the spouses the MMPC alone is not enough so we resort to the MMMB algorithm. Its pseudocode is presented in 2.5. In the forward phase it identifies the PC(C), it then calls the algorithm recursively to find the candidate set of the Markov Blanket (CMB) [13], it considers the the unioin of PC of the features in the set of candidates of parents and children, excluding $C \cup CPC(C)$ as the set of candidate spouses (CSP(C)). Then, for each feature Y in this set,

and for each feature X in the CPC(C) set, it is tested if there is a subset $S \subseteq Fn\{C, X, Y\}$ such that $C \perp\!\!\!\perp Y|S$ and $C \not\perp\!\!\!\perp Y|X \cup S$, Y is considered as a spouse of C, doing this MMMB can distiguish parents from children in order to identify the spouses [7]. To filter the false positives, MMMB tests a property that only true spouses of C have: Conditioned on any subset that includes a common child X and C are dependent [13].

MMPC and MMMB

MMPC(T)

/* add true positives to PC */
1 $PC = \emptyset$
2 **repeat**
3 **for each** $X \in (\mathbf{U} \backslash PC \backslash \{T\})$ **do**
4 $Sep[X] = arg: min_{\mathbf{Z} \subseteq PC} dep(T, X|\mathbf{Z})$
5 $Y = arg: max_{X \in (\mathbf{U} \backslash PC \backslash \{T\})} dep(T, X|Sep[X])$
6 **if** $T \not\perp\!\!\!\perp Y|Sep[Y]$ **then**
7 $PC = PC \cup \{Y\}$
8 **until** PC does not change
/* remove false positives from PC */
9 **for each** $X \in PC$ **do**
10 **if** $T \perp\!\!\!\perp X|\mathbf{Z}$ for some $\mathbf{Z} \subseteq PC \backslash \{X\}$ **then**
11 $PC = PC \backslash \{X\}$
12 **return** PC



MMMB(T)

/* add true positives to MB */
1 $PC = MMPC(T)$
2 $MB = PC$
3 $CanMB = (PC \cup_{X \in PC} MMPC(X)) \backslash \{T\}$
/* add more true positives to MB*/
4 **for each** $X \in CanMB \backslash PC$ **do**
5 find any Z such that $T \perp\!\!\!\perp X|\mathbf{Z}$ and $T, X \notin \mathbf{Z}$
6 **for each** $Y \in PC$ **do**
7 **if** $T \not\perp\!\!\!\perp X|\mathbf{Z} \cup \{Y\}$ **then**
8 $MB = MB \cup \{X\}$
9 **return** MB

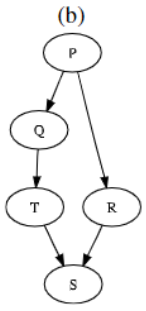


Figure 2.5: Pseudocode of the MMPC and MMMB algorithms [14]

PCMB Parents and children based (PCMB), returns the true MB of a target variable C, in the corresponding DAG, under the assumptions of faithfullness and causal sufficiency. As we can see in the pseudocode in figure 2.6, PCMB makes use of two functions: *GetPCD* and *GetPC* in order to solve the first subproblem previously explained of the divide and conquer approach [14], *GetPCD* to find CPC(C) and *GetPC* to remove false positives in the candidates set [7]. PCMB tries to minimize the number of nodes that don't belong to PC(C) that are returned in a superset of PC(C), PCD, it repeats 3 steps until PCD doesn't change, some nodes that are not in the parents-children set are removed from the candidate set, the candidate that is most likely to be in PC(C) is selected and added to PCD and removed from the candidates set [14], by calculating the association of each feature and C, conditioned on all possible subsets of CPC(C) and chooses the minimum. If X and C assessed as independent, the feature will be removed from CPC(C) [7]. It identifies spouses in a way similar to the previous method, for each feature Y of PC(C) if a subset S within F\{C,X,Y} exists such that $C \perp\!\!\!\perp Y|S$ and $C \not\perp\!\!\!\perp Y|S \cup \{X\}$, then Y is a spouse of C regarding X [7]. It is data efficient since the number of instances necessary to identify the

MB(C) only depend on the topology of the graph and not on the size of the MB [14].

GetPCD, GetPC and PCMB

```
GetPCD(T)
1   PCD = ∅
2   CanPCD = U\{T}
3   repeat
      /* remove false positives from CanPCD */
4     for each X ∈ CanPCD do
5       Sep[X] = arg:min_{Z⊆PCD} dep(T, X|Z)
6     for each X ∈ CanPCD do
7       if T ⫫ X|Sep[X] then
8         CanPCD = CanPCD\{X}
      /* add the best candidate to PCD */
9     Y = arg: max_{X∈CanPCD} dep(T, X|Sep[X])
10    PCD = PCD ∪ {Y}
11    CanPCD = CanPCD\{Y}
      /* remove false positives from PCD */
12    for each X ∈ PCD do
13      Sep[X] = arg: min_{Z⊆PCD\{X}} dep(T, X|Z)
14    for each X ∈ PCD do
15      if T ⫫ X|Sep[X] then
16      PCD = PCD\{X}
17  until PCD does not change
18  return PCD
```

```
GetPC(T)
1   PC = ∅
2   for each X ∈ GetPCD(T) do
3     if T ∈ GetPCD(X) then
4       PC = PC ∪ {X}
5   return PC

PCMB(T)
    /* add true positives to MB */
1   PC = GetPC(T)
2   MB = PC
    /* add more true positives to MB */
3   for each Y ∈ PC do
4     for each X ∈ GetPC(Y) do
5       if X ∉ PC then
6         find Z st T ⫫ X|Z and T, X ∉ Z
7         if T ⫫̸ X|Z ∪ {Y} then
8           MB = MB ∪ {X}
9   return MB
```

Figure 2.6: Pseudocode of the PCMB algorithm [14]

STMB The Simultaneos MB algorithm (STMB) was developed to try to improve the computational efficiency problem, that results of the thorough search for conditioning sets to identify the parents and children of each feature in CPC(C). It does this by following two steps: firstly it identifies the spouse of the class variable C from F\CPC(C), contrarily to PCMB in which the union of parents and children for each feature in CPC(C) was used [7], it searchs for a node,X, in the set of found parents and children, that unblocks a path from C to other node, Y, that belongs to the set of variables that exclude the true parents and children set. If a node Y is found, X could be a spouse [15]. Afterwards, It removes the false positives, the non-MB descendants, from the candidate set using the using the candidate set of spouses instead of the symmetry check used previously, making the STMB possibly more efficient than previous methods. However, STMB is still data inefficient since it uses the entire set as a conditional set [7].

2.5.3 MB learning with interleaving PC and spouse learning

This type of approach, works as an extension of the previous one by implementing the parents and children learning phase and the spouse learning phase alternatively. When a candidate member of the set of the parents and children is added to PC(C) during the PC learning phase, the spouse learning phase set off by this approach straight away [7].

We have two algorithms developed for the method, **BAMB** and **EEMB**. After learning the candidates for PC and SP, BAMB removes the false positives at once from the two candidate sets; contrarily, EEMB breaks BAMB into two subroutines: learning and pruning.

This type of method tries to keep the set of candidates for both the SP and PC set by intervaling the learning of both of these sets in order to achieve a balance between data and time efficiency, however

STMB Algorithm

Input: Data, D; target node, T
$\mathbf{PC}_T \leftarrow \mathbf{V} \setminus \{T\}$;
{step 1: find the PC set }
$[\mathbf{PC}_T, \mathbf{Sep}_T] \leftarrow$ RecogPC(T, $\mathbf{PC}_T$, D);
{step 2: find spouses and remove **non-child** descendants}
$\mathbf{spouse}_T \leftarrow \emptyset$;
$\mathbf{remove} \leftarrow \emptyset$;
for each $Y \in \mathbf{P}C_T$ **do**
  for each $X \in \{\mathbf{V} \setminus \{T\} \setminus \mathbf{PC}_T\}$ **do**
    if $X \not\perp\!\!\!\perp T | \mathbf{Sep}_T\{X\} \cup \{Y\}$ **then**
      if $Y \perp\!\!\!\perp T | \mathbf{Z}, \exists \mathbf{Z} \subseteq \mathbf{PC}_T \cup \{X\} \setminus \{Y\}$ **then**
        $\mathbf{remove} \leftarrow \mathbf{remove} \cup \{Y\}$;
        break;
      else
        $\mathbf{spouse}_T\{Y\} \leftarrow \mathbf{spouse}_T\{Y\} \cup \{X\}$;
      end if
    end if
  end for
end for
$\mathbf{PC}_T \leftarrow \mathbf{PC}_T \setminus \mathbf{remove}$;
for each Y in $\mathbf{spouse}_T$ **do**
  for each X in nonempty $\mathbf{spouse}_T\{Y\}$ **do**
    $\mathbf{testSet} \leftarrow \mathbf{PC}_T \cup \mathbf{spouse}_T\{Y\} \setminus \{X\}$;
    if $X \perp\!\!\!\perp T | \mathbf{testSet}$ **then**
      remove X from $\mathbf{spouse}_T\{Y\}$;
    end if
  end for
end for
$\mathbf{M} \leftarrow \mathbf{PC}_T$;
for each $X \in \mathbf{M}$ **do**
  if $X \perp\!\!\!\perp T | \mathbf{PC}_T \cup \mathbf{spouse}_T \setminus \{X\}$ **then**
    $\mathbf{PC}_T \leftarrow \mathbf{PC}_T \setminus \{X\}$;
  end if
end for
$\mathbf{MB} \leftarrow \mathbf{spouse}_T \cup \mathbf{PC}_T$;

Figure 2.7: Pseudocode of the STMB algorithm [15]

because there might be a few false PC inclusions, a lot of false spouses may enter the SP set which will result in a larger size of the candidate set [7].

BAMB The Balance MB learing algorithm (BAMB) uses the IFBS framework to turn the learning of the PC set and identifying the spouses into one single procedure [7]. This algorithm works in three steps as seen in the pseudocode of figure 2.8. The first step consists in finding the candidates for the PC and SP set [16], when a new feature is added to the set of candidate parents and children, the algorithm starts to find the spouses of C regarding the current feature. It then makes use of the found set SP(C) so it can remove false positives from CPC(C) and uses this last set to prune SP(C).Because of this BAMB keeps the two sets as small as possible to try and achieve both data and time efficiency. Despite this, because of the false positives included in PC set, a lot of false spouses can enter SP(C) which will lead to a search in the union of the current SP(C) and CPC(C) so the false parents and children can be removes,

which can make BAMB not only time but also data inefficient [7].

The BAMB Algorithm

Input: D: dataset; T: target.
Output: $[\mathbf{PC}_T, \mathbf{SP}_T]$: Markov blanket of T.

1. $\mathbf{CPC}_T \leftarrow \emptyset$;
2. $\mathbf{Tmp} \leftarrow \mathbf{U} \setminus \{T\}$;
3. $\mathbf{Sep}_T\{X\} \leftarrow \emptyset$ for each $X \in \mathbf{U} \setminus \{T\}$ and $T \perp\!\!\!\perp X|\emptyset$;
4. **repeat**
   */*Step 1: Find the candidate set of PC and candidate set of spouses*/*
5. $A \leftarrow argmax_{X \in \mathbf{Tmp}} dep(T, X|\emptyset)$;
6. $\mathbf{Tmp} \leftarrow \mathbf{Tmp} \setminus \{A\}$;
7. **if** $T \perp\!\!\!\perp A|\mathbf{Z}$ for some $\mathbf{Z} \subseteq \mathbf{CPC}_T$ **then**
8. $\mathbf{Sep}_T\{A\} \leftarrow \mathbf{Z}$;
9. **else**
10. $\mathbf{CPC}_T \leftarrow \mathbf{CPC}_T \cup \{A\}$;
11. **for** each $B \in \mathbf{CPC}_T$ and $B \neq A$ **do**
12. **if** $T \perp\!\!\!\perp B|\mathbf{Z}$ for some $\mathbf{Z} \subseteq \mathbf{CPC}_T \setminus \{B\}$ **then**
13. $\mathbf{CPC}_T \leftarrow \mathbf{CPC}_T \setminus \{B\}$;
14. $\mathbf{CSP}_T\{B\} \leftarrow \emptyset$;
15. $\mathbf{Sep}_T\{B\} \leftarrow \mathbf{Z}$;
16. **end if**
17. **end for**
18. $\mathbf{CSP}_T\{A\} \leftarrow \emptyset$;
19. **for** each C in $\{\mathbf{U} \setminus \{T\} \setminus \mathbf{CPC}_T\}$
20. **if** $T \not\perp\!\!\!\perp C|\mathbf{Sep}_T\{C\} \cup \{A\}$ **then**
21. $\mathbf{CSP}_T\{A\} \leftarrow \mathbf{CSP}_T\{A\} \cup \{C\}$;
22. **end if**
23. **end for**
   */*Step 2: Remove false positives from the candidate set of spouses*/*
24. $\mathbf{SP}_T\{A\} \leftarrow \emptyset$;
25. **repeat**
26. $E \leftarrow argmax_{X \in \mathbf{CSP}_T\{A\}} dep(T, X|\mathbf{Sep}_T\{X\} \cup \{A\})$;
27. $\mathbf{CSP}_T\{A\} \leftarrow \mathbf{CSP}_T\{A\} \setminus \{E\}$;
28. $\mathbf{SP}_T\{A\} \leftarrow \mathbf{SP}_T\{A\} \cup \{E\}$;
29. **for** each $X \in \mathbf{SP}_T\{A\}$ **do**
30. **if** $T \perp\!\!\!\perp X|\mathbf{Z} \cup A$ for some $\mathbf{Z} \subseteq \mathbf{CPC}_T \cup \mathbf{SP}_T\{A\} \setminus \{X\}$ **then**
31. $\mathbf{SP}_T\{A\} \leftarrow \mathbf{SP}_T\{A\} \setminus \{X\}$;
32. **end if**
33. **end for**
34. **until** $\mathbf{CSP}_T\{A\}$ is empty
   */*Step 3: Remove false positives from the candidate set of PC*/*
35. $\mathbf{PC}_T \leftarrow \emptyset$;
36. **repeat**
37. $F \leftarrow argmax_{X \in \mathbf{CPC}_T} dep(T, X|\emptyset)$;
38. $\mathbf{CPC}_T \leftarrow \mathbf{CPC}_T \setminus \{F\}$;
39. $\mathbf{PC}_T \leftarrow \mathbf{PC}_T \cup \{F\}$;
40. **for** each $X \in \mathbf{PC}_T$ **do**
41. **if** $T \perp\!\!\!\perp X|\mathbf{Z} \cup_{Y \in \mathbf{Z}} \mathbf{SP}_T\{Y\}$ for some $\mathbf{Z} \subseteq \mathbf{PC}_T \setminus \{X\}$ **then**
42. $\mathbf{PC}_T \leftarrow \mathbf{PC}_T \setminus \{X\}$;
43. $\mathbf{SP}_T\{X\} \leftarrow \emptyset$;
44. **end if**
45. **end for**
46. **until** $\mathbf{CPC}_T$ is empty
47. $\mathbf{CSP}_T\{A\} \leftarrow \mathbf{SP}_T\{A\}$
48. $\mathbf{CPC}_T \leftarrow \mathbf{PC}_T$
49. **end if**
50. **until Tmp** is empty

Figure 2.8: Pseudocode of the BAMB algorithm [16]

2.5.4 MB learning with relaxed assumptions

Every approach that was mentioned so far assumes that the faithfullness and causal sufficiency assumptions are met, however when confronted with practical situations this is not always the case. When the faithfullness assumption is not met a Markov Blanket may not be unique, meaning that when researching for it more than one MB may be identified without this assumption. In the case that the causal sufficiency assumption is not true, the Markov Blanket that the algorithm learns may not specify the true causal relationships [7]. The Target Information Equivalence presented in the following paragraph is a great example of this type of approach.

TIE* The Target Information Equivalence (TIE*) is used for multiple MB learning, by relaxing the composition assumption to the local composition assumption, we can check if the joint probability P(V) satisfies this assumption with respect to C, by assuring that $C \perp\!\!\!\perp S_1|Z$ and $C \perp\!\!\!\perp S_2|Z$, $C \perp\!\!\!\perp (S_1 \cup S_2)|Z$ [7].The pseudocode for the steps mentioned next can be seen in figure 2.9. TIE* starts by using a Markov Boundary induction algorithm in order to learn the MB(C) from the given data an outputs the MB to be used in the next step [17], and it then creates a new data set D_n that may help identifying a new MB. This potential MB may have been "invisible" to the previously used algorithm, concealed by another MB. In the last step, the same algorithm used in the beginning is applied to D_n and a new candidate set $CMB_n(C)$ is created. If this new set is also a MB of the original dataset (according to independence tests or classification accuracy) it is set as a new MB of the class variable. These steps are repeated until no new datasets can be generated. This algorithm outputs all possible MBs of a class variable even when the faithfulness assumption is not met [7].

Generative algorithm **TIE***

Inputs:
- dataset D (a sample from distribution P) for variables ***V***, including a response variable *T*;
- Markov boundary induction algorithm X;
- procedure Y to generate datasets from the embedded distributions;
- criterion Z to verify Markov boundaries of *T*.

(specific examples of inputs X, Y, Z are given in subsection 4.2)

Output: all Markov boundaries of *T* that exist in P.

1. Use algorithm X to learn a Markov boundary ***M*** of *T* from the dataset D for variables ***V*** (i.e., in the original distribution P)
2. Output ***M***
3. Repeat
4. Use procedure Y to generate a dataset D^e from the embedded distribution by removing a subset of variables ***G*** from the full set of variables ***V*** in the original distribution (also denoted as $D(\boldsymbol{V} \setminus \boldsymbol{G})$).
5. Use algorithm X to learn a Markov boundary $\boldsymbol{M}_{new}$ of *T* from the dataset D^e
6. If $\boldsymbol{M}_{new}$ is a Markov boundary of *T* in the original distribution according to criterion Z, output $\boldsymbol{M}_{new}$
7. Until all datasets D^e generated by procedure Y have been considered.

Figure 2.9: Pseudocode of the TIE* algorithm [17]

2.6 Score-Based methods

A different type of methods that may be used to discover the Markov Blanket, or the Parents-Children set are Score-Based methods. Instead of resorting to independence tests to do this, these methods make use of score-based BN learning algorithms, which don't focus on the MB learning research and, for that reason, there are a significantly fewer algorithms than in the previous section [7].

When given a dataset D, a score-based method has as goal finding the structure of the BN, represented by the DAG that maximizes a scoring function. This function quantifies the fitness between the DAG and the dataset D and is usually combined with a greedy search method so that from the set of feasible solutions the adequacy of each structure being explored is measured. This problem of learning the BN structure can be formulated as:

> Given D, learning a DAG G* in which G*=$argmax_{G \subseteq O}$ f(G:D) and f(G:D) represents the scoring function and O the family of all possible DAGs defined on D.

It is useful to use aggregate local scores of a DAG in order to get a global score, therefor it is important that the score function is decomposable. This happens when it is possible to express the score assigned to a structure as a combination of local scores of each node and its parents in the graph, $G : f(G : D) = \sum_{V_i \in V} f(V_i; pa_G(V_i) : D_{V_i, pa_G(V_i)})$ [7].

Because scoring functions are indeed decomposable, score-based methods can distinguish parents from children during the MB learning stage, contrary to constraint-based algorithms. This is because score-based methods learn a DAG of features currently selected, C, and a new feature, and next read the MB from the DAG at each iteration [7].

2.6.1 Divide and conquer MB learning

In the case of score based methods, the previously explained approach of dividing the learning of the Markov Blanket in two sub/problems is also commonly used. An example of this type of algorithm is the Score-based Local Learning (SLL) algorithm.

SLL The Score-based Local Learning (SLL) algorithm pseudocode for each phase is shown in Figure 2.10. This algorithm starts by learning the set of parents and children as seen in the first two sections of the pseudocode in Figure 2.10. It initiates the candidate PC set as an empty set, randomly selects a feature at each iteration to remove from the initial feature set, and learns the DAG from the set using a score-based BN learning algorithm. It obtains a new CPC(C) from the DAG. When the feature set is empty, the final candidate set is obtained. Secondly, it prunes CPC(C) using symmetry check and learns the PC set of each feature in the CPC(C) as in the first step. If the class variable does not belong to CPC(C), X is removed from the latter. Next, the spouses of C are identified, and the algorithm finds the union of PC of each feature in PC(C), previously obtained as the candidate spouses of C (CSP(C) $\setminus$ PC(C) $\cup$ C). Next, it learns a DAG for each feature in this set and obtains a new set of spouses of C from the DAG until the union set is empty. Finally, symmetry checks are performed to check the final

spouses, and SLL learns the spouses of all the features. If a spouse set of a certain feature includes C, this feature will be added to the SP(C) set. The process of the symmetry check may be computationally expensive depending on the size of F \ PC(C) [7].

Candidate PC Learning by SLL

Input: Feature set F and the class variable C
Output: $CPC(C)$
$CPC(C) = \emptyset$;
repeat
  Select a feature $X \in F$;
  $F = F \setminus X$;
  //using an existing score-based BN learning algorithm
  Learning a DAG on the set $CPC(C) \cup \{X\} \cup \{C\}$;
  Obtain $CPC(C)$ from the learnt DAG;
until F is empty;
Output $CPC(C)$.

PC Learning with Symmetric Check by SLL

Input: Feature set F and the class variable C
Output: $PC(C)$
Find $CPC(C)$ using Algorithm 9;
//Symmetric check whether $C \in PC(X)(X \in CPC(C))$
repeat
  Select a feature $X \in CPC(C)$;
  $CPC(C) = CPC(C) \setminus X$;
  Obtain $CPC(X)$ using Algorithm 9;
  if $C \notin CPC(X)$ **then**
    $CPC(C) = CPC(C) \setminus X$;
  end if
until $CPC(C)$ is empty;
$PC(C) = CPC(C)$;
Output $PC(C)$.

Spouse Learning by SLL

Input: Feature set F and $PC(C)$
Output: $SP(C)$
Find candidate spouses $CSP(C)$, i.e., PC of each feature in $PC(C)$ using Algorithm 10;
$CSP(C) = CSP(C) \setminus PC(C) \cup C$; $SP(C) = \emptyset$;
repeat
  Select a feature $X \in CSP(C)$;
  $CSP(C) = CSP(C) \setminus X$;
  Learning a DAG on the set $PC(C) \cup \{X\} \cup \{C\} \cup SP(C)$;
  Obtain SP(C) from the learnt DAG;
until $CSP(C)$ is empty;
Output $SP(C)$.

Figure 2.10: Pseudocode of the SLL algorithm [7]

2.6.2 GES

The Greedy Equivalence Search Algorithm (GES) is used to determine a specific sequence of edge modifications to a graph G that transforms it into a graph H in a way that H remains an independence

map of G after each modification.

This is an algorithm that can be divided into two different phases, The Forward Equivalence Search (FES) and the Backward Equivalence Search phase. In the first phase, the algorithm greedily adds edges until a local maximum is reached, this search is performed over equivalence classes using a particular connectivity between these. In the second phase, the edges are greedily removed until a local maximum is reached. This phase proceeds from the previous local maximum using a second connectivity. When the maximum is reached, the equivalent class is returned as a solution. Describing the algorithm more precisely, it starts with an equivalence class that corresponds to no dependencies, and these are greedily added by taking into account all possible single-edge additions that can be made to every DAG in the current equivalent class. As soon as the greedy algorithm stops at a local maximum, the second phase previously mentioned considers all single-edge deletions that can be made to all DAGs at each step. Finally, this algorithm finishes its process and a local maximum is identified at the end of the second phase. The process is detailed in the pseudocode 2.11

The standard implementation of GES considers each variable as a root node in a DAG and adds the parents iteratively so that the score calculated is either maximized or minimized, depending on the goal. This process is repeated until no improvements can be made meaning that the best value of the score was found [23].

```
GES Algorithm:
Input: Data D with variables V
Output: Bayesian network structure G

1. Initialize an empty graph G with nodes corresponding to variables in V.

2. For each pair of variables Xi and Xj in V:
   a. Perform a conditional independence test to determine if Xi and Xj are
      independent given the current graph G and any subset of variables.
   b. If Xi and Xj are dependent given the current graph G, orient the edge
      between them according to the direction that satisfies the conditional independence test.
   c. If Xi and Xj are independent given the current graph G, mark the edge
      as undirected between Xi and Xj.

3. Repeat step 2 until no more edge orientation or marking can be done.

4. Return the final graph G as the Bayesian network structure.

Conditional Independence Test:
Input: Variables Xi, Xj, current graph G, and possibly a subset of variables S.
Output: True if Xi and Xj are conditionally independent given S in the context of G; otherwise, False.

1. For each possible ordering of the variables in V:
   a. Construct a Bayesian network with the proposed edge orientations.
   b. Estimate the conditional probability distribution of Xi given Xj and S using data D and the network.
   c. Check if Xi and Xj are conditionally independent given S using the estimated
      distribution and a statistical test (e.g., a chi-squared test or likelihood ratio test).
   d. If Xi and Xj are found to be conditionally independent, return True.

2. If Xi and Xj are not found to be conditionally independent for any ordering, return False.
```

Figure 2.11: Pseudocode of the GES algorithm

2.7 Methods for distinguishing parents from children

In the previous methods, it was not possible to distinguish parents (direct causes) from children (direct effects), which would be useful for predicting the consequences of interventions in decision-making. In the case in which the training and test data are obtained from different distributions, the set of parents of a class variable can be used as the set of invariant features so that robust predictions can be achieved [7].

2.7.1 Global BN structure learning

This strategy starts by learning the Markov Blanket of each feature using existing methods for causality-based feature selection, it then builds sort of a draft of the DAG, since it starts by being undirected. The edges are later directed using either independence tests or score criteria [7]. Various methods were developed using this type of approach; for example, the GGSL method starts with a randomly selected feature and then expands the learned structure using local learning steps which use a score-based learning algorithm [7]. The PSL algorithm was then developed based on the GGSL but is instead a parallel Bayesian network structure algorithm [7].

2.7.2 Local BN structure learning

Despite the global approach being able to deal with a dataset with thousands of features, in most real-world cases only the causal relationships around only a class variable are of interest, therefore algorithms were developed to learn only what is necessary. These type of algorithms start by finding the PC set of a class variable and build the structure based on this, and then finds the Markov Blanket of the feature connected to the first class variable while building the local structure for each of the features. This is done until all the parents are completely distinguished from the children or until they can not be distinguished any further [7].

Chapter 3

Implementation

3.1 Choosing a toolbox

When starting the research for this work, I had the goal of having the PyCausal toolbox [19] as a base for it. However, after further experiments with this particular tool, I realized that it had a few issues and errors that made it impossible to use it for the designated purpose. Because of this, another toolbox showed to be more adequate and is the base for the practical work developed and described, this is the gcastle toolbox developed by HUAWEI Noah's Ark Lab and available at [20].

3.1.1 GCastle toolbox

The GCastle (Gradient-based causal structure learning) toolbox is an end-to-end Python toolbox that has a GUI implementation to facilitate the visualization of tasks and datasets. This tool includes dataset generation (either simulated or generated based on real-world data), causal discovery, and evaluating the previously learned graphs. Four different types of algorithms are implemented that can be used in the causal discovery functionality: Constraint-based, function-based, score-based, and gradient-based algorithms. This is highly relevant for this work since the goal is to use different scores, used in the score-based methods to evaluate datasets during the causal discovery step. It implements several methods for each category, original PC, stable PC, and parallel PC for constraint-based algorithms, DirectLiNGAM, ICALiNGAM, ANM, and HPCI as the function-based methods, GES (which was discussed in the previous chapter) and TTPM for the score-based methods, and finally GraNDAG, NOTEARS, MCSL, GAE, RL-BIC and CORL for gradient-based methods.

The GUI Implementation of this toolbox also makes it of extreme pertinence and differentiates it from other researched toolboxes by helping with the visualization of each task. It is extremely easy to create and manage a task either for data generation or causal discovery since a list of tasks exists, with a number, name, progress bar, and several options to edit or visualize each task. Besides, it makes it possible not only to visualize the discovered graph and compare it to the true one but also to evaluate the discovered graph using 9 different metrics (FDR, TPR, FPR, SHD, NNZ, precision, recall, F1, and score).

Finally, the toolbox has two different scores implemented and ready to use in continuous variables (whose generation is also already implemented), the BIC score and the BDeu score. The goal is to use these implementations as base work and develop firstly a function to generate categorical data, and secondly, variations of the already implemented scores in order to evaluate the outcome of doing causal discovery on different graphs. These variations will consist on the base BIC and BDeu scores, but applied to categorical data, a score that has the entropy value summed to the base score and finally a score that sums the Kullback-Leibler divergence to either the BIC or BDeu score calculated.

3.2 Scores implemented for continuous variables

3.2.1 BIC Score

The Bayesian Information Criterion (BIC) is a widely used tool in statistical model selection [21]. When faced with the problem of to choosing the most suitable model from a collection of viable candidates, BIC presents an asymptomatic approximation to a transformation of the Bayesian posterior probability of a candidate model. The general formulation of BIC is based on the empirical log-likelihood and does not require the specification of priors [21]. The model chosen is used to describe a group of response measurements (y) that are generated as specified by an unknown density g(y). This density is the true generation model, formulated to describe the data, and y is identified as the candidate approximating model. The goal is to find the model that better approximates the true generation model, g(y).

If we denote the likelihood corresponding to the density $f(y|\theta_k)$ as $L(\theta_k|y)$ and the vector of estimates obtained by maximizing the likelihood over Θ_k as $\hat{\theta_k}$, the BIC score can be defined in 3.1. We compute BIC for each of the candidate models and select the model that corresponds to the lowest value of BIC.

$$BIC = -2\ln L(\theta_k|y) + k\ln(n) \tag{3.1}$$

One of the reasons that makes the use of BIC very advantageous is the property of consistency, which means that the probability that the model with the correct structure will be selected approaches one [21]. This model is known as the *quasi-true* model, and it's the most parsimonious model that, as measured by the Kullback-Leibler information, is the closest to the true model. This property is one of the reasons why BIC outperforms other popular methods of model selection, such as AIC [21].

The gcastle toolbox implemented two different BIC scores to use with continuous variables.

BIC by R^2 The first variant of the BIC implemented is the one that uses the R^2 statistic. It calculates the score for a specific y given a set of parent variables using the R^2 statistic which is often used when dealing with linear regression models. It assesses how well the independent variables (the parents) explain the variances in the dependent variable.

This method starts by prepping the data, creating a column vector (Y) from the dataset containing the values of "y" for observations. Then the number of parent variables in the parent set pa_y (k) is

calculated. The method then proceeds to model fitting, if k=0, meaning that there are no parent variables, the model is assumed a simple linear regression with only an intercept term (which means that there are no independent variables). In this specific case, the Residual Sum of Squares (RSS) is calculated as the Total Sum of Squares (TSS) since there are no predictors to explain the variance in ”Y”. If there are parent values (k¿0) then a linear regression model is fitted to predict ”y” based on the parent variables in pa_y. The RSS is computed as the sum of squared differences between the observed values of ”Y” and the predicted values from the linear regression model. Afterward, The R^2 is calculated. This can also be named the coefficient of determination and it measures the proportion of the variation in the dependent variable ”Y” that is explained by the parent variables. This value can be calculated using 3.2. In this context, RSS is the residual sum of squares that was previously calculated and TSS is the sum of squared differences between each data point and the mean of ”Y”.

$$R^2 = 1 - \frac{RSS}{TSS} \tag{3.2}$$

Finally, the BIC score is calculated as stated in 3.3 where n represents the number of observations (sample size), k the number of parent variables (size of pa_y), and R^2, the statistic calculated previously.

$$BIC = n \cdot \log(R^2) + (k+1) \cdot \log(n) \tag{3.3}$$

The BIC score calculated is a measure of the trade-off between model fit, represented by R^2, and model complexity, represented by k. Models with more parameters are penalized unless they improve the fit, meaning having a higher R^2 The pseudocode that summarizes this model is represented in figure 3.1

```
FUNCTION _bic_by_r2(y, pa_y):
# Data Preparation
Y = Extract column 'y' from the dataset
k = Count of parent variables in 'pa_y'

# Model Fitting
IF k == 0:
    # No parent variables, simple linear regression with intercept only
    SSR = Sum of squared differences between Y and its mean
ELSE:
    # Fit a linear regression model with parent variables in pa_y
    X = Extract columns in 'pa_y' from the dataset
    Fit linear regression model Y ~ X
    SSR = Residual sum of squares from the regression model

# Calculate R-squared (coefficient of determination)
TSS = Total sum of squares of Y (sum of squared differences between Y and its mean)
R_squared = 1 - (SSR / TSS)

# Calculate BIC Score
n = Number of observations in the dataset
BIC = n * log(R_squared) + (k + 1) * log(n)

RETURN BIC
```

Figure 3.1: Pseudocode of the bic_by_r2 function

BIC by scatter The second implementation computes the BIC Score given a target variable "Y" and its parents (Pa) using a scatter matrix-based formula. It uses as parameters the index of the target variable "Y" (y) and a list of indexes representing the parent variable of target variables (pa_y) The first step is to compute the scatter matrix for the dataset, it is a square matrix that encapsulates the variances and covariances of the dataset variables. The matrix contains specific information on the spread and relationships among all variables. The method uses the function *np.cov* that calculates the covariance matrix for the dataset. Then, the variance of the target variable is calculated and denoted as *sigma*, this variable corresponds to the diagonal element in the scatter matrix of the target variable. After this, the parent variables are handled, if there are parent variables specified for Y a modified variance is calculated, by considering the effect of these parent variables. Besides the computation of the variance, a partial scatter matrix is computed for the parent variables. This matrix (pa_cov) contains information on the spread and relationships among the parent variables and it's extracted from the scatter matrix. Afterward, the covariance between the target variable and the parent variables is calculated (y_cov). Using the variables computed previously, such as the partial scatter matrix and the covariance the method calculates coefficients that modify *sigma* of the target variable. These coefficients are obtained by solving a linear system of equations. Finally, the BIC score is calculated using the modified variance and the number of parent variables, as seen in 3.4

$$BIC = -(n * (1 + log(sigma)) + (k + 1) * log(n)) \tag{3.4}$$

The pseudocode that resumes the computation of the BIC score using a scatter method is presented in figure 3.2

3.2.2 BDeu

The Bayesian Dirichlet Equivalent Uniform score (BDeu) is a particular case of the Bayesian Dirichlet score (BD), it is used in Bayesian network structure learning and it quantifies how good of a fit a BN model is to a given dataset while penalizing complex models, the key behind this score is to strike a balance between these two things. It is the only one of the BD scores that takes the same value for DAGs in the same equivalent class [22].

One variant of the BDeu is implemented by gcastle, to be used with continuous variables. The specific implementation of the linear BDeu score starts by calculating the configuration of the parent set 'pa_y' obtaining unique values for each parent variable and uses "cartesian_combination" in order to generate all possible combinations of these values. Each combination represents a possible configuration of the set of parents. Then, the two variables used to accumulate the terms in the BDeu linear formula, are initialized and the code enters a loop so it can iterate over all possible parent configurations of the parent set (j_conf), for each configuration. In this loop, n_ij is calculated, representing the number of records when the target variable 'X_i' equals 'k' and the parent set (Pa_i) is the j^{th} configuration. Afterward, the first term (term2) is computed based on the logarithm of gamma functions and involves the structure prior. Next, it recovers the unique states of X_i and initializes 'term3' which will accumulate the

```
Function _bic_by_scatter(y, pa_y):
Input:
y: Index of the target variable Y
pa_y: List of indices representing parent variables Pa of Y
Output:
    bic_score: Bayesian Information Criterion (BIC) score

# Calculate the scatter matrix for the dataset
scatter = ComputeScatterMatrix(data)

# Calculate the variance of the target variable Y (sigma)
sigma = scatter[y, y]

# Handle parent variables (Pa) if present
if len(pa_y) > 0:
    pa = List(pa_y)
    k = Length(pa)

    # Compute the partial scatter matrix of Pa variables
    pa_cov = scatter[pa, :][:, pa]

    # Compute the covariance between Y and Pa variables
    y_cov = scatter[y, pa]

    # Solve for the coefficients using the partial scatter matrix and covariance
    coef = SolveLinearEquation(pa_cov, y_cov)

    # Calculate the adjusted variance of Y
    sigma = sigma - y_cov @ coef

# Calculate the BIC score
bic_score = - (n * (1 + log(sigma)) + (k + 1) * log(n))

Return bic_score
```

Figure 3.2: Pseudocode of the bic_by_scatter function

second term in the BDeu linear formula. Then the code enters another loop in order to iterate over all the possible states of X_i (k) and calculates the value of n_ijk for each state 'k', which consists of the number of records in which the target variable equals 'k'. The second term (term3) is computed similarly to the first, using the logarithmic of the gamma functions and involving the equivalent sample size prior. Finally, the calculated "term3" is added to the accumulative "term1" for the current parent configuration and the BDeu score is computed by summing "term0" and "term1". This score represents the BDeu score for the given parent set and target variable. This score is computed for each possible parent configuration and the final BDeu score for a specific set of parents is the sum of the scores for all possible configurations. The pseudocode for this process is represented in figure 3.3.

3.3 Scores implemented for categorical variables

The previously mentioned implementations only work in continuous data. In order to score causal discovery processes for categorical data, a few adaptations were necessary in the previous code. A version of BIC and another of BDeu were implemented to work with categorical data.

```
Function bdeu_linear(y, pa_y):
# Calculate configurations of the parent set pa_y
conf_pa_y = generate_parent_configurations(pa_y)

# Get the cardinality of variable y
r_i = get_cardinality(y)

# Calculate q_i, the product of cardinalities of parent variables in pa_y
q_i = calculate_q_i(pa_y)

# Initialize term0, which corresponds to the structure prior
term0 = (r_i - 1) * q_i * log(k)

# Initialize term1, which will accumulate the terms involving equivalent sample size
term1 = 0

# Loop over all possible parent configurations
for j_conf in conf_pa_y:
    # Calculate Nijk, the number of records where X_i = k and Pa_i is in j_conf
    n_ij = calculate_nijk(y, pa_y, j_conf)

    # Calculate the first term in the BDeu linear formula
    term2 = lgamma(N / q_i) - lgamma(n_ij + N / q_i)

    # Get the unique states of variable X_i
    states_i = get_unique_states(y)

    # Initialize term3 for accumulating terms for different states of X_i
    term3 = 0

    # Loop over all states of X_i
    for k in states_i:
        # Calculate Nijk for state k
        n_ijk = calculate_nijk(y, pa_y, j_conf, k)

        # Calculate the second term in the BDeu linear formula
        term3 += (lgamma(n_ijk + N / (r_i * q_i))
                  - lgamma(N / (r_i * q_i)))

    # Add term2 and term3 to term1 for the current parent configuration
    term1 += (term2 + term3)

# Calculate the final BDeu score
bdeu_linear = term0 + term1

# Return the BDeu score for the given parent set pa_y and target variable y
return bdeu_linear
```

Figure 3.3: Pseudocode of the bdeu_linear function

3.3.1 Function to generate categorical data

In order to test different types of data, it was necessary to create a specific function to generate categorical data. This function starts by defining a softmax function, to be used later on to calculate the probabilities required to sample the categorical variables. This function computes the softmax of an input array. Afterward, it extracts information about the SEM such as the number of variables(d) and the number of unique categories (num_categories) in the array of categories that is an input to the function. Next, a DAG is created (G_nx), a representation of the SEM, using the library *NetworkX*. This

structure is used to represent the causal relationships between the different variables in the SEM. The function then performs a topological sorting of the DAG in order to determine the order in which the variables should be generated. This ensures that no errors occur because of a variable that depends on another that comes later in the order. The data matrix 'X' is initialized with shape (n,d) to store the generated data the shape means that the matrix will have the number of samples as number of rows and variables as columns. Next, the function encodes the categories using scikit-learn's 'LabelEncoder. This is necessary since machine learning models typically only work with numerical data, and the encoder maps each unique category to a unique numerical label. Finally, the function iterates through each variable 'j' in the topological order. If it's independent (has no parent variables) it samples the categories uniformly for all the number of samples. If it has parent variables, it calculates a weighted sum of these using weights from the matrix 'W'. It then applies the softmax function previously computed to the weighted sum to obtain a probability distribution over the categories, and it samples 'n' categories from this distribution and assigns them to 'j' in the data matrix 'X'. This is the final matrix that is returned.

As done for the previous functions, the pseudocode in figure 3.4 summarizes the process previously described.

3.3.2 BIC for categorical data

This function starts as the previous ones by preparing the data, meaning that numpy matrix representation of the data set is assigned to the data matrix variable. Afterward, variable Y extracts the value of the target variable from this matrix and the number of parent variables is computed, and stored in variable k to be used later in the process. The log-likelihood is then calculated depending on the value of k. If there are no parent variables (k=0), the log-likelihood is computed for a univariate distribution using the gamma function, by summing the logarithmic of the gamma function applied to each data point in Y. Besides, it also subtracts the logarithm of the gamma function applied to the total number of data points in order to finally compute the log-likelihood. In the case of the value of k being bigger than zero (there are parent variables) the log-likelihood is calculated based on the configurations of the parent variables. It starts by constructing X, which represents the values of the parent variables for each data point, and then counts the occurrences of each unique parent configuration and the combined occurrences of the parent-variable configurations. After this, the log-likelihood can be computed. Using the calculated value of the log-likelihood, we can compute the value of the BIC score using variables k, and self.n (representing the total number of data points). This process is summarized in pseudocode 3.5

There are four main differences between this implementation and the previous ones used in continuous data: the modeling approach, the way the log-likelihood is calculated, how data types are handled, and how the parameters are counted. The modeling approach, in the case of the categorical approach, assumes that the target and parent variables are categorical and handles the probability distributions and respective counts accordingly. The calculation of the log-likelihood is also different since it is estimated using the values of the computed probabilities and gamma function. In the case of how this model handles the data type, it assumes that the data for the target are categorical and it performs operations

```
Function _simulate_categorical_sem(n, categories, W, noise_scale, sem_type)
num_categories = length(unique(categories))
d = number of variables
function softmax(x)
    exp_x = exp(x)
    return exp_x / sum(exp_x)

Create an empty directed acyclic graph G_nx

for each edge (i, j) in W
    Add directed edge from variable j to variable i in G_nx

if G_nx is not a directed acyclic graph
    Raise an error: "W must be a DAG"

ordered_vertices = TopologicallySort(G_nx)

Initialize an empty data matrix X of size (n, d)

Encode categorical categories using LabelEncoder to get encoded_categories

repeated_categories = Repeat encoded_categories n//len(encoded_categories) times
remaining_elements = n % len(encoded_categories)
if remaining_elements > 0
    Concatenate the first remaining_elements of encoded_categories to repeated_categorie
s

for each variable j in ordered_vertices
    parents = Parents of variable j in G_nx

    if length(parents) == 0
        # Independent variable, sample categories uniformly
        for each sample i from 1 to n
            X[i, j] = RandomChoice(repeated_categories)

    else
        # Dependent variable
        for each sample i from 1 to n
            parent_values = Values of parent variables in X for sample i
            weighted_sum = DotProduct(parent_values, W[parents, j])
            probs = softmax(weighted_sum)
            X[i, j] = RandomChoice(repeated_categories, probabilities=probs)

return X
```

Figure 3.4: Pseudocode of the function that generates categorical data

accordingly. Finally, in the categorical case, the variable k represents the number of parents, contrary to the continuous case in which usually it represents the number of model parameters.

3.3.3 BDeu for categorical data

Adapting the previous BDeu function, it's possible to use a new function to compute the value of the score for categorical data, as seen in the pseudocode in figure 3.6. It calculates the score for a given node y and its parent configurations pa_y in the context of a Bayesian Network. The function starts, similarly to the linear BDeu, by initializing the matrix representation of the stored data, the data matrix variable, the r_i variable which is the number of unique states of the node y, obtained from the r_i_map,

```
function _bic_by_categorical(self, y, pa_y)
data_matrix = self.data.values  # Convert the dataframe to a numpy matrix
Y = data_matrix[:, y]  # Extract the target variable Y
k = length(pa_y)  # Calculate the number of parent variables
if k == 0:
    # If there are no parent variables, calculate the log likelihood directly
    log_likelihood = sum(log(gamma(Y + 1))) - log(gamma(self.n + 1))
else:
    pa = list(pa_y)  # Create a list of parent variable indices

    X = data_matrix[:, pa]  # Extract the parent variable columns
    pa_counts = apply_along_axis(tuple, axis=1, arr=X)  # Count parent configurations
    pa_config_counts = unique(pa_counts, return_counts=True, axis=0)  # Count parent configurations

    # Combine parent configurations with target variable values
    combined_counts = concatenate(pa_counts, Y)

    # Count the number of occurrences of each combined configuration
    combined_config_counts = unique(combined_counts, return_counts=True, axis=0)

    # Compute the log likelihood using the combined configurations
    log_likelihood = compute_log_likelihood(combined_config_counts[1], pa_config_counts[1])

# Calculate the BIC score using the log likelihood and the number of parameters (k)
bic_score = log_likelihood - 0.5 * k * log(self.n)

print("number of samples:", k, "log likelihood:", log_likelihood, "BIC score:", bic_score)

return bic_score
```

Figure 3.5: Pseudocode of the function that computes the BIC score for categorical data

and finally the q_i that uses the q_i_cat(pa_y) method, that calculates the product of the parent nodes in pa_y. Next, the term0 is computed, it represents the penalty for adding a parent configuration to the node y. It's calculated using equation 3.5, where the number of states in the categorical value is given by k. The term1 is initialized to be used later in order to accumulate values in the following steps.

$$term0 = (r_i - 1) * q_i * \ln(k) \tag{3.5}$$

The function then iterates over all the possible parent configurations (j_conf) for the node y. These are obtained using the method _get_parent_configuration(pa_y). For each of these configurations, the count of occurrences n_ij, of the specific configuration in the data is computed,term2 is calculated as the difference between the logarithmic of two gamma functions, as seen in equation 3.6, where N is the number of data points.

$$\ln(N/q_i) - \ln(n_ij + N/q_i) \tag{3.6}$$

Finally, the function iterates over values of k, and for each of these states, it calculates n_ij and j_conf (the count of occurrences and different configurations respectively) and term3 accumulates the logarithmic differences of gamma functions for each k. With this, term1 can accumulate the values of term2 and term3 calculated previously for all parent configurations, and the final BDeu score is computed as the sum of term0 and term1.

```
function bdeu_cat(y, pa_y):
data_matrix = GetDataMatrix()  # Matrix representation of the data
r_i = GetCardinality(y)  # Cardinality of node y
q_i = CalculateProductOfParentCardinalities(pa_y)  # Product of parent cardinalities
term0 = (r_i - 1) * q_i * log(k)  # Term 0 calculation
term1 = 0  # Initialize Term 1

for j_conf in GetParentConfigurations(pa_y):
    n_ij = CalculateConfigurationCount(y, pa_y, j_conf)  # Count of j_conf
    term2 = ln(N / q_i) - ln(n_ij + N / q_i)  # Term 2 calculation
    term3 = 0  # Initialize Term 3

    for k in GetUniqueStatesOfY():
        n_ijk = CalculateStateCount(y, pa_y, j_conf, k)  # Count of j_conf with state k
        term3 += ln(n_ijk + N / (r_i * q_i)) - ln(N / (r_i * q_i))  # Term 3 accumulation

    term1 += term2 + term3  # Accumulate Term 2 and Term 3

bdeu_score = term0 + term1  # Calculate BDeu score

Print("BDeu Score for Node y:", bdeu_score)  # Print BDeu score

return bdeu_score  # Return BDeu score
```

Figure 3.6: Pseudocode of the function that computes the BIC score for categorical data

3.3.4 Scores using entropy

In order to compare how different scores behave when dealing with categorical data, a new score was implemented. This score uses the different probabilities of parent configurations to compute the entropy and sum it to the previously computed scores.

BIC score using entropy This method needs several other functions to compute different values that will be useful in computing the final entropy value. Firstly a function to calculate the unique combined configurations and their respective counts is needed, it counts the occurrences of each parent-variable configuration for a give y, its parent set pa and a specific value v. It does this by extracting the relevant columns from the data set based on y and pa, and combined them. Then, another function counts unique parent configurations for a given parent set (pa_y), it handles the cases in which there are no parents, single parents, or multiple parents. In the case of single parents, the unique values and respective counts are returned, and in the case of multiple parents, the parent data frame is converted to tuples and counts the unique parent configurations. To finalize the functions needed before entering the main function, the values of the conditional probability for a given variable y which takes a value v, given a specific configuration c.

After this, the BIC score using the entropy is computed, first, it is checked if there are parents, when that is not the case, the log-likelihood, entropy, and final score are computed directly from the dataset, if parent values exist, the log-likelihood and entropy are calculated using parent configurations and combined configurations in a loop over unique parent configurations. Finally, the final score is computed using the expression in 3.7, in which alpha regulates the weight of the entropy value in the final score. The pseudocode for this is represented in figure 3.7.

$$BICScore = LogLikelihood - \frac{1}{2}k\ln(n) + \alpha \cdot entropy \tag{3.7}$$

```
Function _bic_by_entropy(y, pa_y):
Input:
- y: Index of the target variable
- pa_y: List of parent variable indices
Output:
    - bic_score: Bayesian Information Criterion (BIC) score

Constants:
    - alpha: Regularization term

Variables:
    - entropy: Initialize to 0
    - Y: Extract the values of the target variable y from the dataset
    - k: Number of parent variables in pa_y
    - epsilon: A small constant (e.g., 1e-5) for numerical stability
    - log_likelihood: Initialize to 0

If pa_y is empty:  # No parent variables
    Compute log_likelihood based on gamma function:
        - log_likelihood = Sum of gammaln(Y + 1) - gammaln(total_samples + 1)
    Compute entropy based on target variable values:
        - entropy = -Mean(Y * log(Y + epsilon))
    Compute BIC score based on log_likelihood, entropy, and alpha:
        - bic_score = log_likelihood - 0.5 * log(total_samples) + alpha * entropy
Else:
    Iterate over unique parent configurations (c) and their counts:
        Compute conditional probability p_y_given_pa:
            - p_y_given_pa = compute_probability(y, pa_y, Y, c) + epsilon
        If p_y_given_pa contains no zeros:
            Update log_likelihood:
                - log_likelihood += Count * log(p_y_given_pa)
            Update entropy:
                - entropy += -p_y_given_pa * log(p_y_given_pa + epsilon)
    Compute combined_counts based on parent configurations and target variable value
s
    Compute bic_score based on log_likelihood, entropy, k, and alpha
    Sum bic_score values
    Write bic_score to an output file
    Write parent configurations to another output file
    Print number of samples (k), log_likelihood, and bic_score
Return bic_score
End Function
```

Figure 3.7: Pseudocode of the function that computes the BIC score using the entropy value

BDeu score using entropy Similarly to the BIC score, the BDeu score was also adapted to use the value of the entropy. It uses similar functions as used in the case of the BIC score to compute the number of possible parent combinations, of what are the unique configurations, and respective counts, and the conditional probability that corresponds to the target variable having a certain value given a specific configuration. The computation of the score itself uses all these values by, firstly checking the length of pa_y which defines if parent variables exist. In the case in which we don't have parent variables, the entropy is calculated based directly on the target variable values, in the opposite case in which

there are parent variables, the compute_probability function is used so that the probabilities for various configurations are computed and the entropy is calculated based on these values. Then, the BDeu score is computed using the previous function bdeu_cat which simply computes the value of the bdeu score for categorical values, and finally, the entropy is summed to this value. This score uses a lot of functions previously mentioned and described and the pseudocode of the main function that computes the final score can be seen in figure 3.8

```
FUNCTION bdeu_entropy(y, pa_y):
# Initialize entropy and a small value epsilon to avoid log(0)
entropy = 0
epsilon = 1e-5
# Extract target variable values
Y = data[:, y]

IF pa_y is not empty:
    # Calculate entropy based on parent configurations
    pa_configurations, pa_counts = count_parent_configurations(pa_y)

    # Iterate over each parent configuration c
    FOR EACH configuration c IN pa_configurations:
        # Calculate the probability of Y given the parent configuration c
        p_y_given_pa = compute_probability(y, pa_y, Y, c)

        # Add a small epsilon to avoid log(0)
        p_y_given_pa = p_y_given_pa + epsilon

        IF any(p_y_given_pa != 0):
            # Update the entropy with the contribution from this configuratio
n
            entropy = entropy - (p_y_given_pa * log(p_y_given_pa + epsilon))

# Calculate the BDeu score
bdeu_score = bdeu_cat(y, pa_y) + entropy

IF pa_y is empty:
    # Calculate a simplified BDeu score when there are no parent variables
    entropy = calculate_entropy_based_on_Y(Y)
    bdeu_score = bdeu_score + entropy

RETURN bdeu_score
```

Figure 3.8: Pseudocode of the function that computes the BDeu score using the entropy value

3.3.5 Scores using KL divergence

Kullback-Leibler divergence The Kullback-Leibler (KL) divergence is a statistical measure used to quantify how close a probability distribution p is to another distribution q. It is defined as

$$D_{KL}(p||q) = \sum_i p_i \log_2 \left(\frac{p_i}{q_i} \right). \tag{3.8}$$

It's a non-negative value that reaches zero only if the two distributions are identical [24] and may even be potentially infinite. This divergence is also described as the "coding penalty" since it selects distribution q to approximate the true distribution p [25].

When there's an infinite number of measurements, the KL divergence relates to the average likelihood 3.9 since if p and q are the same $DKL = 0; \text{if} L = 0, DKL = \infty$.

$$D_{KL}(p||q) = -\log_2(L). \tag{3.9}$$

The KL divergence measures the average likelihood of observing infinite data with distribution p if the model q generated the data [25]. The KL divergence is overall an extremely versatile measure used to assess the dissimilarity between probability distributions and has numerous applications in statistics and information theory [25].

Scores using KL divergence In order to compare different approaches to computing the BIC and BDeu scores, the KL divergence was used in a similar way that the entropy value was used in the previous implementation. In order to compute the KL divergence, the probabilities computed and used in bic_by_entropy and bdeu_entropy functions must be sorted in descending order. To do so, the values of p_y_given_pa are appended to the values of the configuration c and the Python function "sorted" to achieve the desired order for the probabilities array. Then the divergence between the first probability and all the others in the sorted_probabilities array, the result is the average KL divergence. Finally, the BIC or BDeu score is computed using the _bic_by_categorical or bdeu_cat respectively and the value of the average KL divergence is summed so that the final score is found. The process and code details to the case of the BIC score can be found in 3.9.

```
function sorting_probabilities(y, pa_y, p_y_given_pa, c):
probabilities = []
probabilities.append((p_y_given_pa, c))
sorted_probabilities = sort(probabilities, key=lambda x: x[0], reverse=True)
return sorted_probabilities[0][0]

function KL_divergence(y, pa_y, sorted_probabilities):
if sorted_probabilities is a float:
return 0.0
else:
first_prob_distribution = sorted_probabilities
kl_divergence_sum = 0
num_distributions = length(sorted_probabilities) - 1
for (probability, parent_config) in sorted_probabilities[1:]:
kl_divergence = calculate_entropy(first_prob_distribution, probability)
kl_divergence_sum += kl_divergence
average_kl_divergence = kl_divergence_sum / num_distributions
return average_kl_divergence

function _bic_divergence(y, pa_y):
epsilon = 1e-5
Y = get_data_column(y)
k = length(pa_y)
if k == 0:
    log_likelihood = calculate_log_likelihood(Y)
    average_kl_divergence = 0.0
else:
    pa = list(pa_y)
    pa_tuple = tuple(pa)
    pa_configurations, pa_counts = count_parent_configurations(pa_tuple)
    log_likelihood = 0

    for (c, count) in zip(pa_configurations, pa_counts):
        p_y_given_pa = compute_probability(y, pa_tuple, Y, c)
        p_y_given_pa = p_y_given_pa + epsilon
        log_likelihood += count * log(p_y_given_pa + epsilon)

    data_matrix = get_data_matrix()
    X = extract_columns(data_matrix, pa)
    pa_counts = count_occurrences(X)
    combined_counts = combine_counts(pa_counts, Y)
    log_likelihood = calculate_log_likelihood(combined_counts)
    sorted_probabilities = sorting_probabilities(y, pa_y, p_y_given_pa, c)
    average_kl_divergence = KL_divergence(y, pa_y, sorted_probabilities)
    write_to_parent_config_file(pa, y)

bic_score = calculate_bic_score(y, pa_y, average_kl_divergence)

if k != 0:
    write_to_bic_scores_file(bic_score)

return bic_score
```

Figure 3.9: Pseudocode of the function that computes the BIC score using the KL divergence

Chapter 4

Results

In order to facilitate a comprehensive comparison of the various scores implemented, three distinct groups of causal graphs were used, each representing different levels of complexity. These groups were designed to encompass a range of scenarios commonly encountered in causal inference. The groups are as follows:

- **Group 1:** Graphs with three nodes and three edges.
- **Group 2:** Graphs with four nodes and three edges.
- **Group 3:** Graphs with four nodes and five edges.

Three unique graphs were created within each group, each characterized by distinct noise scales. The noise scales employed were 1, 10, and 100. The specific graphs used are represented in Figure 4.1. This diversity of graph structures and noise levels allowed for a comprehensive evaluation of the implemented scores under various conditions. In Table 4.1 the edges (meaning the causal connections) of each graph are represented. If we have a connection that goes from node 0 to node 1, it is represented in this chapter by [0]1. If we have several parents, for example if 0 is a child of nodes 1 and two that is represented by [1,2]0.

Beginning with the evaluation of BIC Scores, each of the graphs presented in Figure 4.1 will be used to test the Causal Discovery method chosen (GES) with the different scores implemented (categorical, entropy, and divergence). The three groups of graphs will be used to compare the behavior of each score not only when different sizes are used, but also different noise scales, within each group.

G1_1	G1_10	G1_100	G2_1	G2_10	G2_100	G3_1	G3_10	G3_100
[1]0 [1]2 [2]0 [1, 2] 0	[0]2 [1]0 [1]2 [0,1] 2	[1]0 [1]2 [2]0 [1, 2] 0	[0]1 [0]3 [1]2	[1]0 [1]3 [2]3 [1, 2] 3	[0]1 [2]0 [2]3	[0]1 [0]2 [1]2 [3]1 [3]2 [0, 3] 1 [0, 1, 3] 2	[0]1 [1]2 [3]0 [3]1 [3]2 [0, 3] 1 [1, 3] 2	[0]2 [1]0 [1]2 [3]0 [3]1 [0, 1] 2 [1, 3] 0

Table 4.1: List of edges for each graph used to test

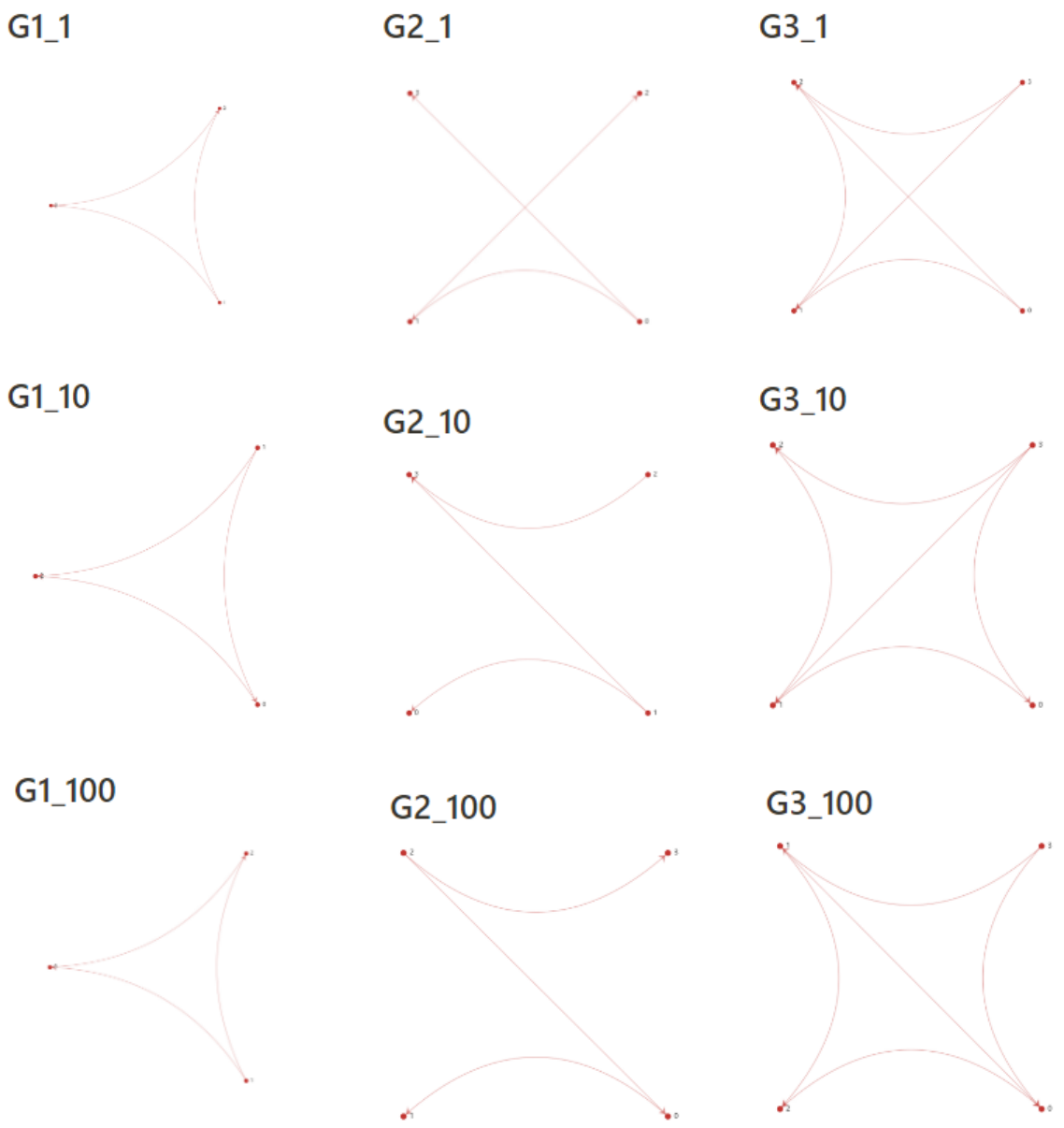


Figure 4.1: Graphs used to test the different scores

Starting with G1_1, the first of the graphs with 3 nodes and 3 edges, within the categorical approach, we observe that [1]0 and [2]0 are correctly identified as causative candidates. However, the complexity of the problem reveals itself as [0]1 and [0]2 are mistakenly identified as causal connections. Besides, it fails to discover [1]2 as a causative link. The different values of this score are shown in Figure 4.2. This graph as well as all the graphs represented in the Figures of this chapter, show the behavior of each score. This means that when analysing one of these graphs we can scrutinize which edges correspond to the highest scores. These graphs, when complemented with the graphical interface of the GCastle toolbox enlighten us on which edges are identified using each score.

Shifting the focus to the entropy-based BIC score, whose values are shown in Figure 4.3, a distinct

pattern emerges. It adeptly uncovers [1]0 and [1]2, showcasing its competence in recognizing these causal relationships. However, it falters in the case of [0]1 and [2]1, misidentifying them as contributing factors. Additionally, [2]0 remains undiscovered by the entropy score, leaving a gap in the causal narrative.

The divergence-based BIC score presents a different perspective. It successfully reveals [1]0 and [2]0 as causal connections, as seen in Figure 4.4, demonstrating its capability to recognize them. Yet, it encounters challenges with [0]1 and [0]2, labeling them incorrectly. Notably, [1]2 remains elusive in the realm of divergence.

When assessing G1_10, meaning that the noise scale is bigger with the categorical approach, it initially mistakenly identifies [2]1 as a causal connection. However, it correctly identifies [1]2 and [1]0 as potential causative links. This all can be seen in Figure 4.8. Unfortunately, [0]1 is wrongly identified, and it fails to discover [1]2 as a causative relationship.

Shifting our focus to the entropy-based BIC score that is represented in Figure 4.9, it also starts by making errors by identifying [2]0 and [2]1 as edges in the estimated graph, which are not present in the true DAG. Nevertheless, it adeptly uncovers [1]2 and [0]2, showcasing its competence in recognizing these causal relationships. However, it leaves [0]2 undiscovered, creating a gap in the causal narrative.

The divergence-based BIC score faces its own set of challenges with [2]0 and [2]1, mislabelling them. It does successfully reveal [1]0 and [2] as causal connections, demonstrating its capability to recognize them. However it also never identifies [0]2 as an edge. This can be seen in Figure 4.10.

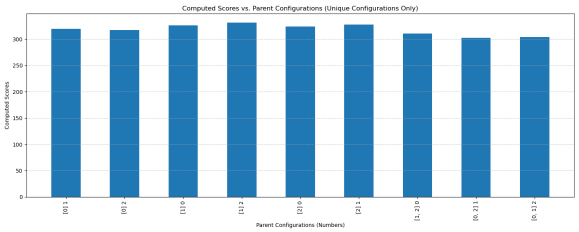


Figure 4.2: BIC categorical G1_1

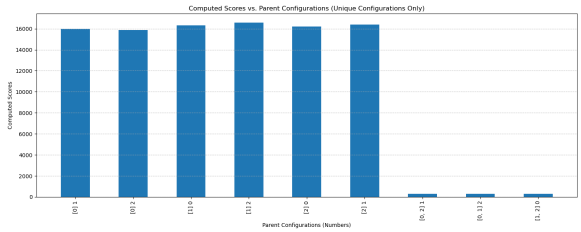


Figure 4.3: BIC entropy G1_1

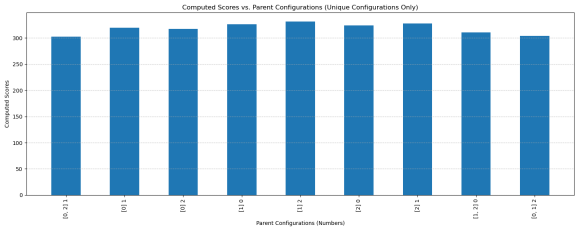


Figure 4.4: BIC divergence G1_1

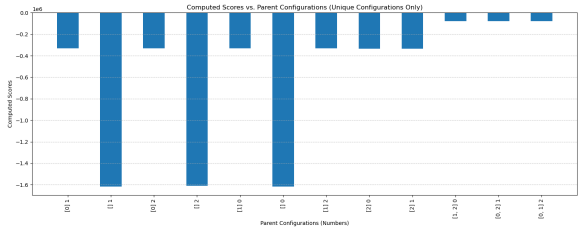


Figure 4.5: BDeu categorical G1_1

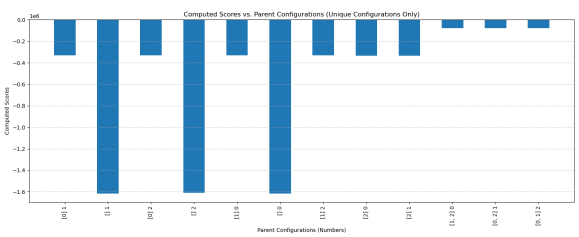


Figure 4.6: BDeu entropy G1_1

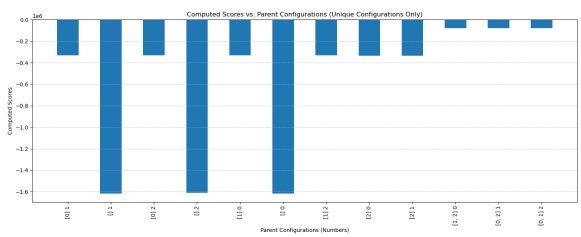


Figure 4.7: BDeu divergence G1_1

In the examination of G1_100, the maximum noise scale of the first group, using the categorical approach, we find that [2]0 and [1]0 are correctly identified as causative candidates. However, the intricacy of the problem becomes evident as [0]1 and [0]2 are mistakenly labeled as causative connections. Unfortunately, it falls short of discovering [1]0 as a causative link.

Turning our attention to the entropy-based BIC score, a different pattern unfolds. It adeptly reveals [2]0 and [1]0, demonstrating its competence in recognizing these causal relationships. However, it stumbles when dealing with [0]1 and [0]2, incorrectly identifying them as contributing factors. Moreover, [1]2 remains undiscovered by the entropy score, leaving a gap in the causal narrative.

The divergence-based BIC score offers a distinct perspective. It successfully identifies [2]0 and [1]0 as causal connections, showcasing its ability to recognize them. However, it faces challenges with [0]1 and [2]1, marking them incorrectly. Notably, [1]0 remains elusive in the realm of the divergence-based approach.

The information referent to the different tests using G1_100 is present in Figures 4.14, 4.15 and 4.16.

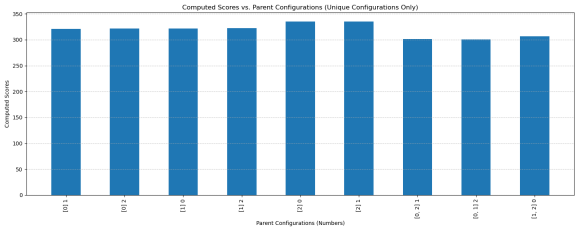


Figure 4.8: BIC categorical G1_10

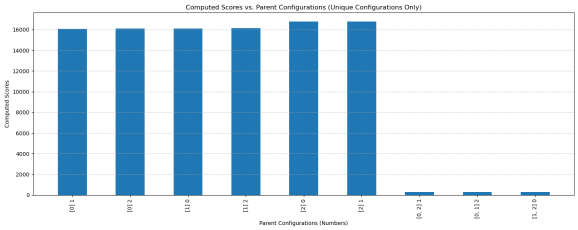


Figure 4.9: BIC entropy G1_10

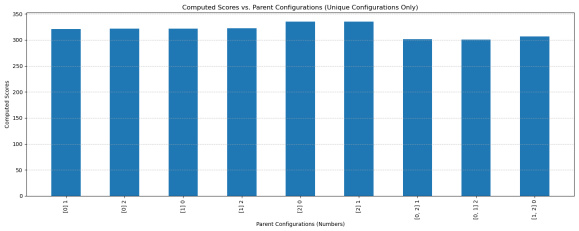


Figure 4.10: BIC divergence G1_10

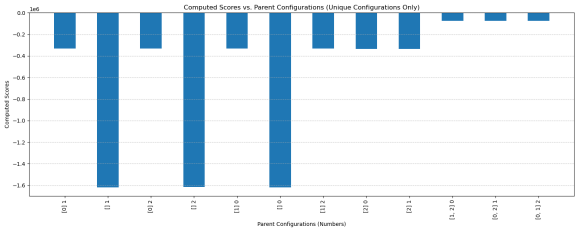


Figure 4.11: BDeu categorical G1_10

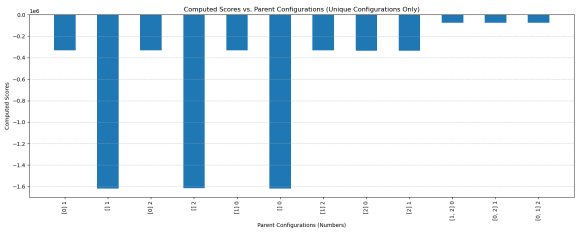


Figure 4.12: BDeu entropy G1_10

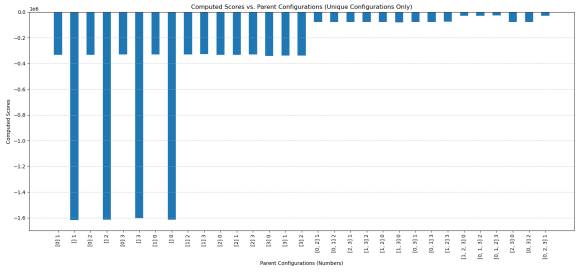


Figure 4.13: BDeu divergence G1_10

Analysing the graphs with 4 nodes and 3 edges, we start with G2_1, which has a noise scale of one.

In the particular case of this graph, when using the categorical approach, two of the correct edges are discovered, [0]1 and [0]3. However, after this only wrong edges are identified, [0]2,[1]0,[3]0, and [2]0. This means that the edge [1]2, that goes from node 1 to node 2 is never discovered. In the particular case of G2_1, the edges discovered are the same in all of the different scores, either using categorical, entropy, or divergence we always get [0]1,[0]3,[3]0,[2]0,[1]0,[0]2 as the discovered edges. This can all be consulted in Figures 4.20, 4.21 and 4.22 The differences are in how many times an edge is scored, meaning that while the times that [0]1 and [0]3 are scored are the same in all the scores, the wrong edges are scored a lot more times in the divergence case.

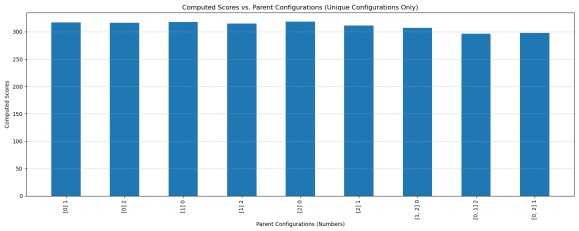


Figure 4.14: BIC categorical G1_100

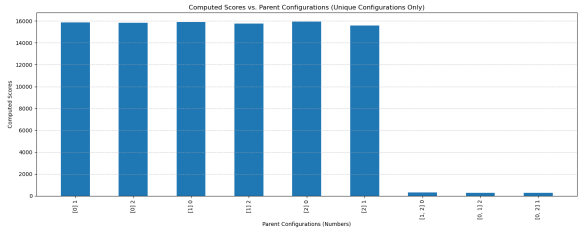


Figure 4.15: BIC entropy G1_100

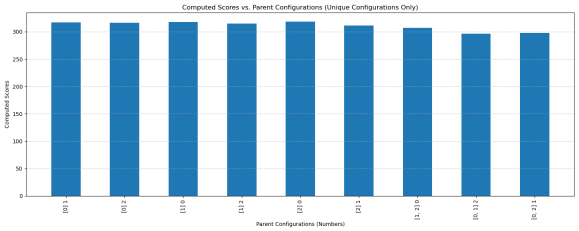


Figure 4.16: BIC divergence G1_100

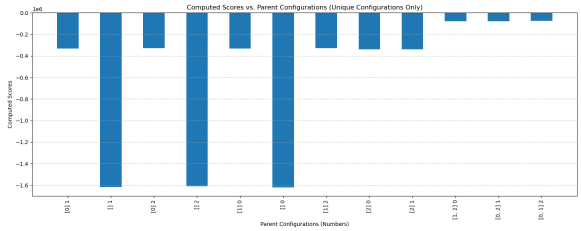


Figure 4.17: BDeu categorical G1_100

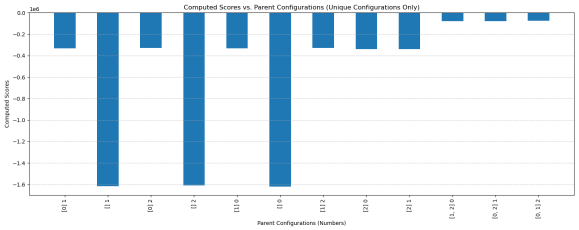


Figure 4.18: BDeu entropy G1_100

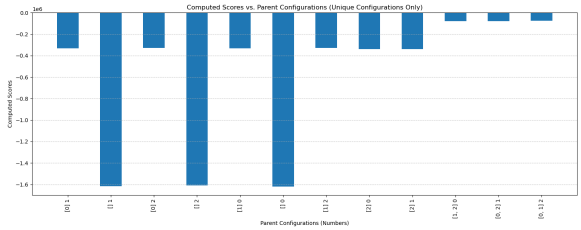


Figure 4.19: BDeu divergence G1_100

When scrutinizing G2_10 through the categorical approach, we discover that [1]3 and [2]3 are accurately pinpointed as causative candidates. Nevertheless, the complexity of identifying causal connections is revealed as [3]2, [3]0, [3]1, and [0]3 are erroneously attributed as causative connections. Regrettably, it falls short of detecting [1]0 as a causative link.This values are all shown in 4.26

Turning our attention to the entropy-based BIC score, whose values are shown in Figure 4.27, it starts by wrongly identifying [1]2 and [2]1 as edges, only after identifying [1]3 as the correct edge. However, after this only wrong connections are identified, respectively [2]0, [3]1,[0]2. This obviously means that using entropy, the [1]0 edge is never found.

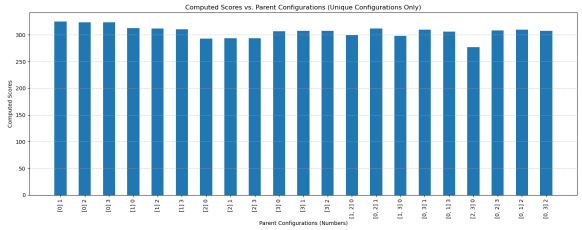


Figure 4.20: BIC categorical G2_1

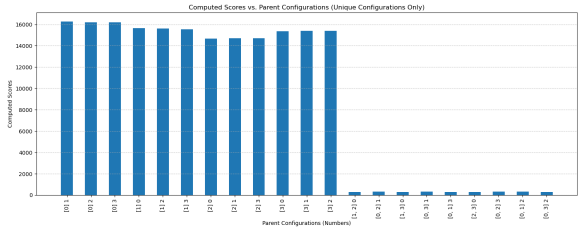


Figure 4.21: BIC entropy G2_1

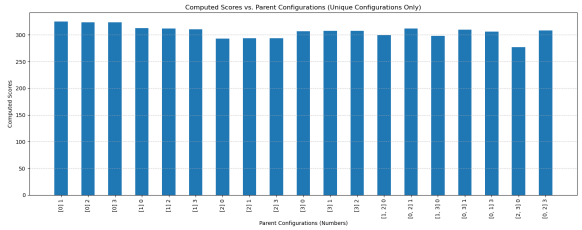


Figure 4.22: BIC divergence G2_1

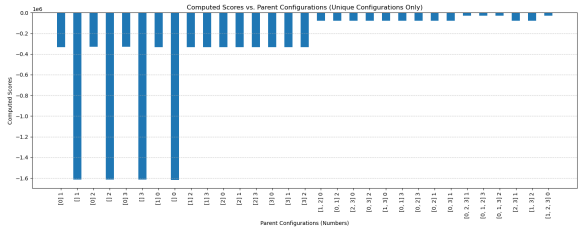


Figure 4.23: BDeu categorical G2_1

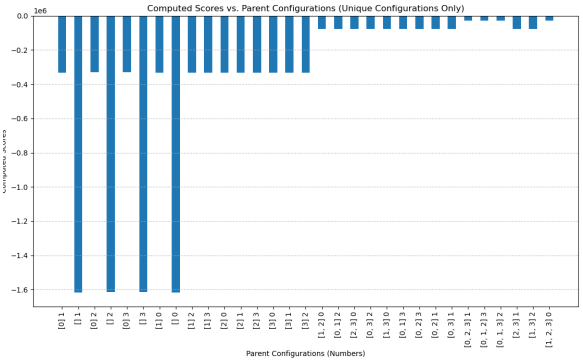


Figure 4.24: BDeu entropy G2_1

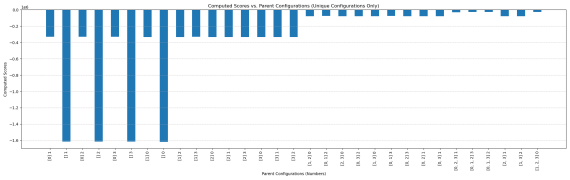


Figure 4.25: BDeu divergence G2_1

The divergence-based BIC score starts by successfully identifying [2] 3 and [1] 3 as causal connections, showcasing its ability to recognize them. However, similarly to the categorical case, [3]2, [3]0, [3]1, and [0]3 are discovered as edges wrongfully, since they're not part of the original graph. This again means that the edge that connects node 1 to 0 is never discovered. This is presented in Figure 4.28.

In the case of G2_100, we have a similar situation to G2_1, meaning that the edges discovered are the same in all of the scores: categorical, entropy, and divergence. The only difference is, again how "fast" the edges are found, all methods start by wrongfully finding [1]2,[1]3, and [1]0 as edges, then the only correct edge, [0]1 is found, and afterward [2]1 and [3]1 are found. In the case of divergence, all of the wrong edges are "found" a lot more times than in the previous scores. All these conclusions can be drawn from the GUI implementation, and Figures 4.32, 4.33 and 4.34.

Finally, testing the graphs with 4 nodes and 5 edges (meaning group 3), starting with G3_1 and G3_10, when comparing the different outcomes of using the categorical, entropy and divergence score, we realize that the causal relations found are the same in the three cases, the wrong edges [1]0, [1]3

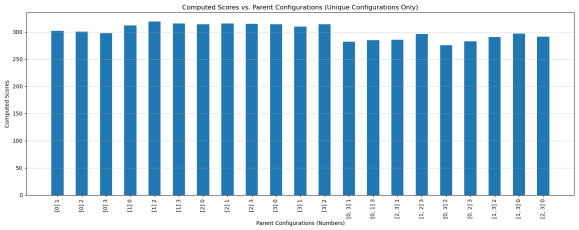


Figure 4.26: BIC categorical G2_10

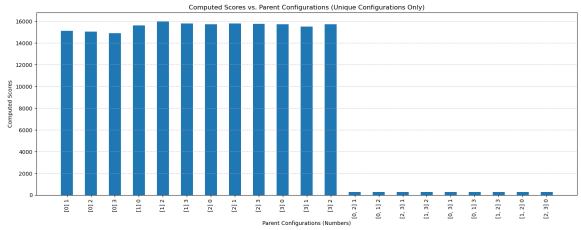


Figure 4.27: BIC entropy G2_10

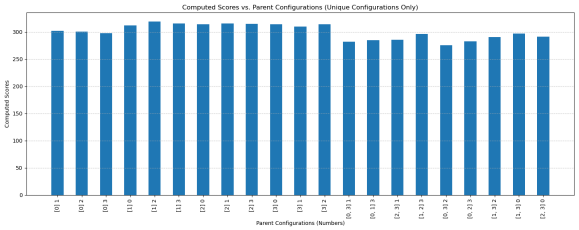


Figure 4.28: BIC divergence G2_10

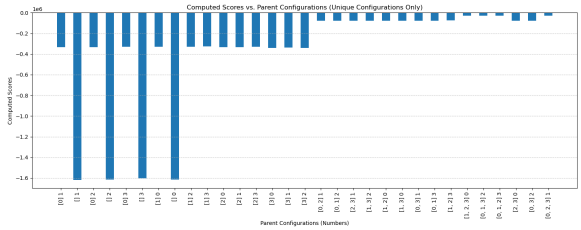


Figure 4.29: BDeu categorical G2_10

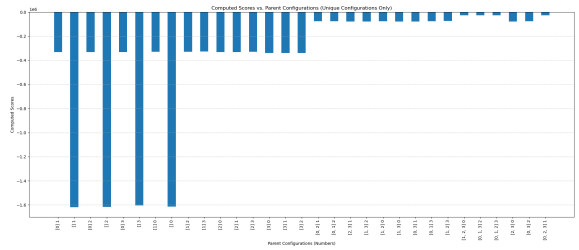


Figure 4.30: BDeu entropy G2_10

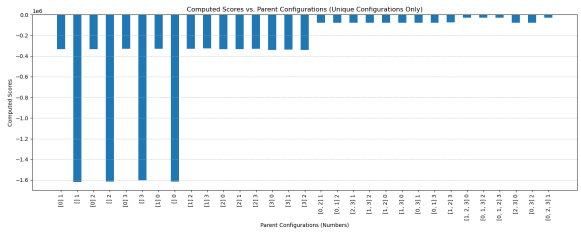


Figure 4.31: BDeu divergence G2_10

and [2]1 are found and only afterward [1]2, [0]1 and finally [3]1. In the case of using entropy the wrong edges are given a score more times than in the categorical case, but still not as many times as in the divergence case.

In the conclusive examination of G3_100 with the categorical score, we encounter a sequence where two incorrect edges, [2]1 and [1]3, are identified before the true edges, [1]2 and [3]1, come to light. Following this, another erroneous edge, [3]2, is mistakenly considered a causal connection ahead of a true edge, [3]0. Lastly, a misjudged edge, [0]3, is revealed. Unfortunately, the remaining edges, [0]2 and [1]0, remain undiscovered throughout the assessment.

When using the entropy-based score, different edges are discovered, with fewer values being calculated. It also starts by revealing three incorrect edges, [2]1, [2]3, and [2]0. It then identifies the correct causal relation of [1]2 before considering the wrong edge [3]2. Finally, the correct edge [0]2 is discovered. This means that the edges [1]0, [3]0, and [3]1 are not discovered.

In the concluding evaluation employing the divergence score, the assessment commences with the identification of incorrect edges, specifically [2]3 and [1]3, before correctly recognizing the edge [3]1. Subsequently, the mistaken inclusion of [3]2 as a causal connection precedes the acknowledgment of the true [3]0. To wrap up, a spurious causal link, [0]3, is brought into consideration. In this case, edges

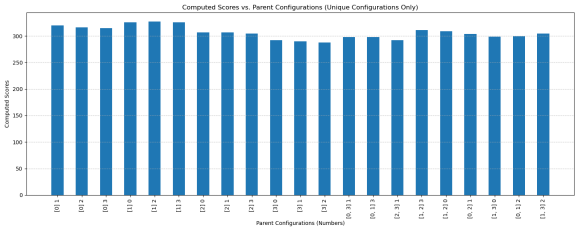


Figure 4.32: BIC categorical G2_100

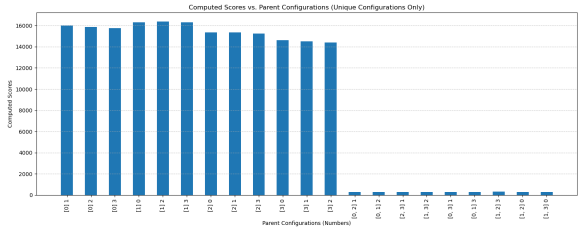


Figure 4.33: BIC entropy G2_100

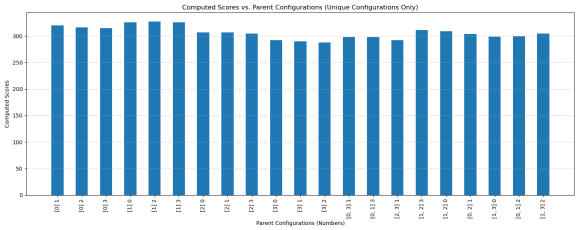


Figure 4.34: BIC divergence G2_100

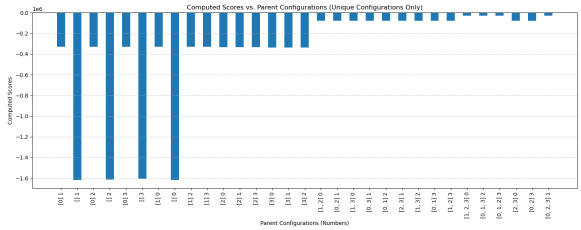


Figure 4.35: BDeu categorical G2_100

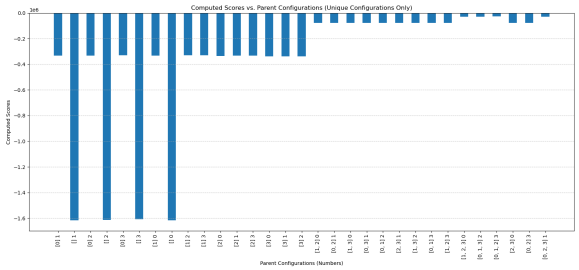


Figure 4.36: BDeu entropy G2_100

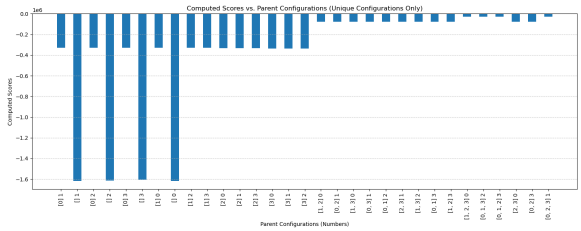


Figure 4.37: BDeu divergence G2_100

[0]2, [1]0, and [1]2 are never considered. These values are presented through Figures 4.38 to 4.52.

Proceeding to the evaluation of the different BDeu scores, starting again with the graph G1_1, when testing the three variants of the BDeu score (categorical, entropy, and divergence) we realize that all of the true edges are found, however, the edges of the opposite causal connection are also discovered. This means that after finding [1]2, [2]1 is also found, after finding [1]0 and [2]0, [0]1 and [0]2 are also discovered. The difference between the three scores is simply the order in which the edges are discovered.

This same behavior can be seen in G1_10 and G_100, again with only the order in which the edges are discovered changing when using different scores. These conclusions can be drawn from Figures 4.5 to 4.19.

Transitioning to the second group of graphs, after analyzing not only the GUI Implementation but also Figures 4.23 to 4.37, we can draw a few conclusions. We commence with G2_1. In this scenario, all the same edges are disclosed, with the only correct ones being [0]1 and [1]2. However, it's important to note that during the process of uncovering these two correct edges, eight incorrect edges are considered. The fundamental difference across the various scores lies in the frequency with which a score is assigned to

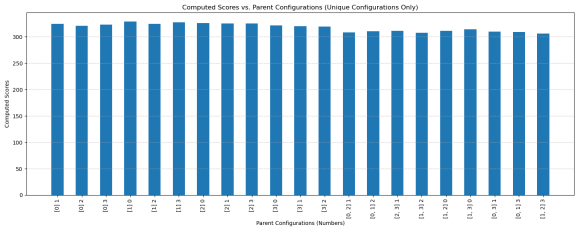


Figure 4.38: BIC categorical G3_1

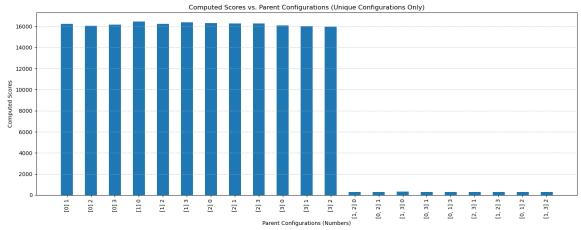


Figure 4.39: BIC entropy G3_1

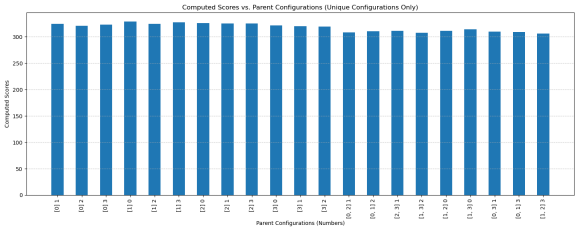


Figure 4.40: BIC divergence G3_1

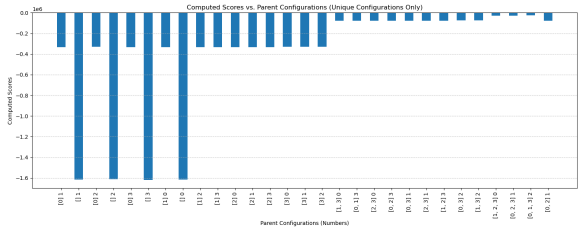


Figure 4.41: BDeu categorical G3_1

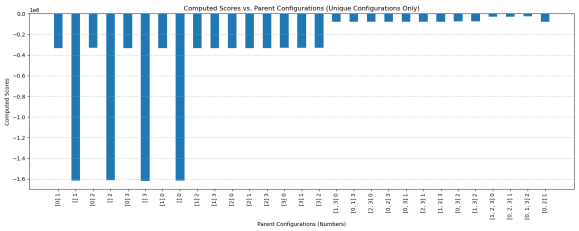


Figure 4.42: BDeu entropy G3_1

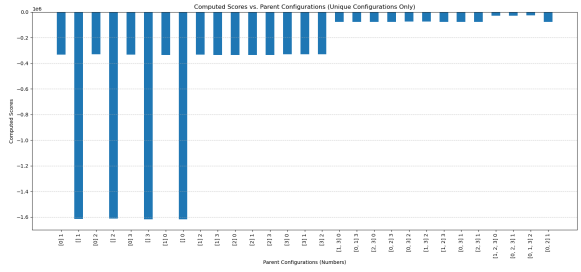


Figure 4.43: BDeu divergence G3_1

a particular configuration.

This is the same thing that happens with G2_10, where only two correct causal connections are discovered while 8 wrong ones are uncovered, and to G2_100 which unveils all the correct edges at the cost of discovering 7 wrong causal connections using all three different BDeu scores.

In the case of the final group of graphs, after testing using the BDeu scores a similar graph to the second group is generated. This can be seen in Figures 4.41 to 4.55. This means that in each of the cases, the BDeu scores find all of the true causal relations, however, to do it they also consider an inappropriate amount of wrong edges that will make it difficult to draw any conclusions on what the real DAG is. After analyzing all these results, it can be concluded that the BIC derivates of the 3 different scores may be a better bet in order to get to a conclusion as to what the real causal relations are in the true DAG. However, it is important to note that when increasing the number of nodes and edges of a graph, the scores tend to not only discover not so many of the correct relations but also wrongly identify more edges. This can also be concluded when analyzing what happens when the noise scale is bigger.

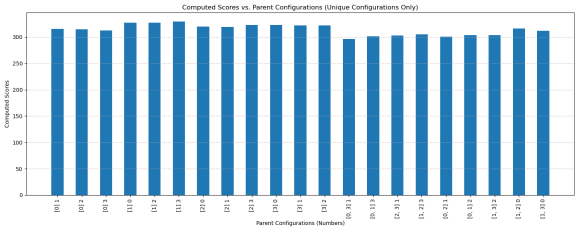


Figure 4.44: BIC categorical G3_10

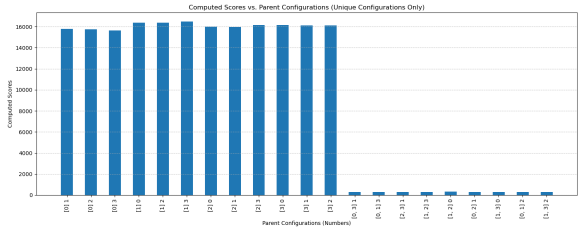


Figure 4.45: BIC entropy G3_10

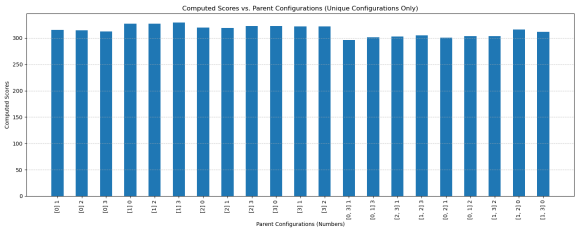


Figure 4.46: BIC divergence G3_10

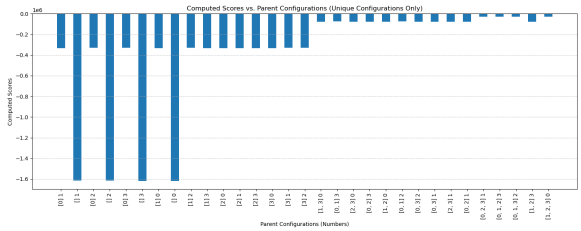


Figure 4.47: BDeu categorical G3_10

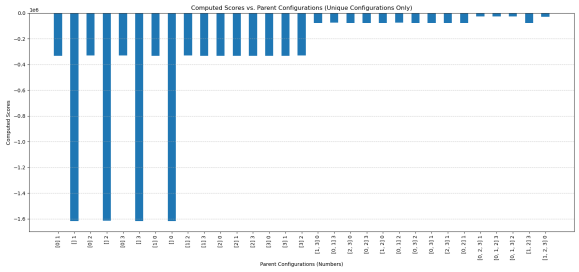


Figure 4.48: BDeu entropy G3_10

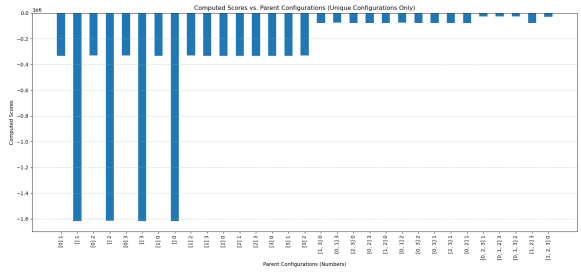


Figure 4.49: BDeu divergence G3_10

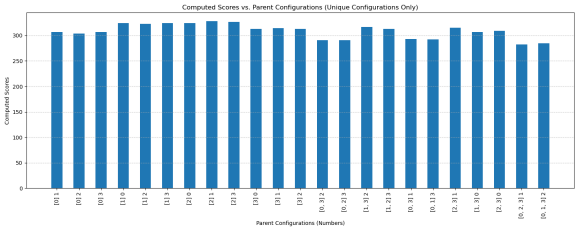


Figure 4.50: BIC categorical G3_100

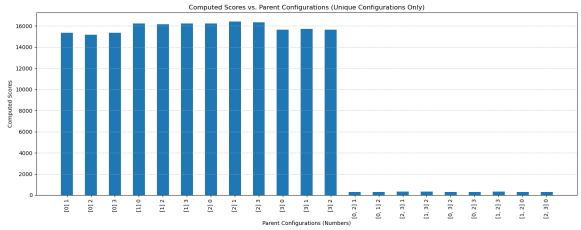


Figure 4.51: BIC entropy G3_100

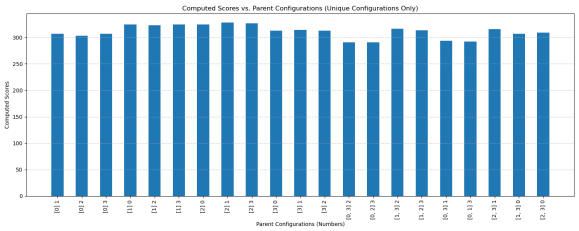


Figure 4.52: BIC divergence G3_100

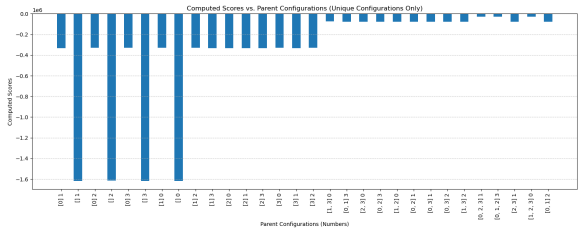


Figure 4.53: BDeu categorical G3_100

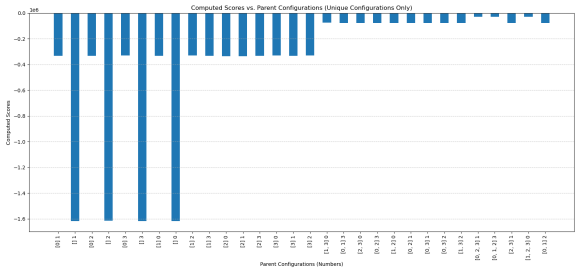


Figure 4.54: BDeu entropy G3_100

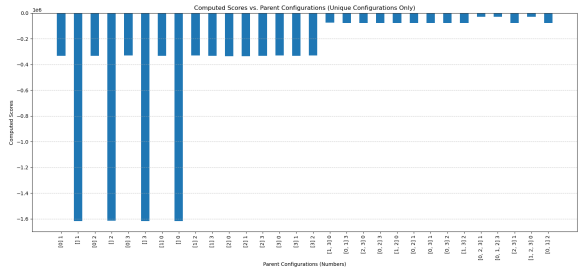


Figure 4.55: BDeu divergence G3_100

Chapter 5

Conclusions

In the course of this thesis, several methods used to infer causal relations were analyzed. The goal was to implement new scores, based on pre-existing ones in order to compare their respective outcomes. In this final chapter, conclusions are drawn and suggestions for future work are made.

5.1 Contributions

We started with a brief introduction on causal inference, a fundamental topic in machine learning and data analysis that poses a significant challenge, as it seeks to uncover the hidden causes and effects within complex systems. At the heart of this challenge lies the intricate web of relationships between variables, often eluding simple observation and demanding sophisticated methods to disentangle. Throughout this thesis, we navigated this complex landscape by exploring a few of the methods designed to tackle the problem of causal discovery. These methods were categorized into three main groups: constraint-based, score-based, and those that distinguish parents from children in causal relationships. We chose the GES (Greedy Equivalence Search) algorithm to serve as the foundation for our experiments. It allowed us to scrutinize different scoring methods across various graph structures and with varying levels of noise. Among the scoring methods, we focused on two key groups: the Bayesian Information Criterion (BIC) and the Bayesian Dirichlet equivalent uniform (BDeu) score, each possessing three distinct variants: the standard categorical score, an entropy-based score, and a score rooted in Kullback-Leibler (KL) divergence. The results of our comprehensive investigation provided a detailed view of causal inference. We uncovered that BDeu, in its quest for causation, often cast a wide net, discovering an abundance of edges. While this may seem promising at first glance, it poses a challenge in distinguishing true causal connections from spurious ones. BIC, on the other hand, emerged as a more precise scorer, making it easier to identify genuine causal links. Furthermore, our exploration unveiled the impact of graph size and noise scale on the performance of these scores. The balance between graph complexity and the level of noise added a layer of intricacy to the causal discovery process. It became apparent that the choice of score should be made with careful consideration of these contextual factors.

5.2 Future Work

The research presented in this thesis has not only advanced our understanding of causal discovery and Bayesian network structure learning but has also illuminated several directions for future research in this evolving field.

First and foremost, future work can focus on Algorithm Refinement. While this study has meticulously evaluated existing methods, there is still room for improvement and innovation. One promising avenue is the development of hybrid algorithms that draw from the strengths of various scoring methods. This approach could potentially elevate the accuracy and robustness of causal structure learning.

Another critical challenge that awaits exploration is Handling Heterogeneous Data. Real-world datasets are often composed of a mix of data types, including continuous and categorical variables. Future research can be dedicated to the development of algorithms capable of effectively handling this data heterogeneity, thus bridging the divide between different scoring techniques.

Lastly, the real-world impact of causal discovery extends to numerous domains, with particular relevance in healthcare, economics, and social sciences. Applications in healthcare and beyond hold significant promise. Further research in these areas may unlock valuable insights, enabling data-driven decision-making and enhancing our understanding of complex systems.

In conclusion, this thesis contributes a foundational understanding of scoring methods in causal discovery. It not only provides a framework for assessing these methods but also illuminates the path ahead. By pursuing algorithm refinement, addressing the challenges of heterogeneous data, and exploring practical applications, researchers can unlock the full potential of causal discovery methods and their relevance across diverse domains. The future of causal discovery research is both exciting and promising, offering numerous opportunities to advance the field and generate insights that drive informed decision-making.

Bibliography

[1] J. Pearl, Causality. Cambridge: Cambridge University Press, 2009.

[2] J. Peters, D. Janzing, and Scholkopf Bernhard, Elements of causal inference: Foundations and learning algorithms. Cambridge, MA: The MIT Press, 2017.

[3] I. Guyon, A. R. Statnikov, and B. B. Batu, Cause effect pairs in machine learning. Cham, Switzerland: Springer, 2019.

[4] J. M. Mooij, J. Peters, D. Janzing, J. Zscheischler, and B. Schölkopf, "Distinguishing cause from effect using observational data: methods and benchmarks," arXiv [cs.LG], 2014.

[5] J. Peters, D. Janzing, and B. Schölkopf, "Causal inference on discrete data using additive noise models," IEEE Trans. Pattern Anal. Mach. Intell., vol. 33, no. 12, pp. 2436–2450, 2011.

[6] J. Peters, J. Mooij, D. Janzing, and B. Schölkopf, "Causal discovery with continuous additive noise models," arXiv [stat.ML], 2013.

[7] K. Yu et al., "Causality-based feature selection: Methods and evaluations," ACM Comput. Surv., vol. 53, no. 5, pp. 1–36, 2021.

[8] P. O. Hoyer, J. Mooij, J. Peters, and B. Schölkopf, "Nonlinear causal discovery with additive noise models," Nips.cc. [Online]. Available: https://papers.nips.cc/paper/2008/file/f7664060cc52bc6f3d620bcedc94a4b6-Paper.pdf. [Accessed: 02-Jan-2023].

[9] "The do-calculus revisited Judea Pearl keynote lecture," Ucla.edu. [Online]. Available: https://ftp.cs.ucla.edu/pub/stat_ser/r402.pdf. [Accessed: 02-Jan-2023].

[10] D. Margaritis and S. Thrun, Bayesian network induction via local neighborhoods. 2000. In Advances in neural information processing systems. 505–511.

[11] I. Tsamardinos and C. Aliferis, Towards principled feature selection: Relevancy, filters and wrappers. 2003.

[12] G. Borboudakis and I. Tsamardinos, "Forward-backward selection with early dropping," arXiv [cs.LG], 2017.

[13] I. Tsamardinos, C. F. Aliferis, and A. Statnikov, “Time and sample efficient discovery of Markov blankets and direct causal relations,” in Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’03, 2003.

[14] J. M. Peña, R. Nilsson, J. Björkegren, and J. Tegnér, “Towards scalable and data efficient learning of Markov boundaries,” Int. J. Approx. Reason., vol. 45, no. 2, pp. 211–232, 2007.

[15] T. Gao and Q. Ji, “Efficient Markov blanket discovery and its application,” IEEE Trans. Cybern., vol. 47, no. 5, pp. 1169–1179, 2017.

[16] Z. Ling, K. Yu, H. Wang, L. Liu, W. Ding, and X. Wu, BAMB: A balanced Markov blanket discovery approach to feature selection. 2019.

[17] A. Statnikov, N. I. Lytkin, J. Lemeire, and C. F. Aliferis, “Algorithms for discovery of multiple Markov boundaries,” J. Mach. Learn. Res., vol. 14, pp. 499–566, 2013.

[18] Bradyneal.com. [Online]. Available: https://www.bradyneal.com/Introduction_to_Causal_Inference-Aug27_2020-Neal.pdf. [Accessed: 02-Jan-2023].

[19] AITiger, PyCausalFS: pyCausalFSA Python Library of Causality-based Feature Selection for Causal Structure Learning and Classification.

[20] Huawei Noah’s Ark Lab. (2023). TrustworthyAI. GitHub. https://github.com/huawei-noah/trustworthyAI/tree/master/gcastle

[21] A. A. Neath and J. E. Cavanaugh, “The Bayesian information criterion: background, derivation, and applications,” Wiley Interdisciplinary Reviews: Computational Statistics, vol. 4, no. 2, pp. 199–203, Dec. 2011, doi: 10.1002/wics.199.

[22] M. Scutari, “Dirichlet Bayesian network scores and the maximum relative entropy principle,” Behaviormetrika, vol. 45, no. 2, pp. 337–362, Apr. 2018, doi: 10.1007/s41237-018-0048-x.

[23] D. M. Chickering, “Optimal Structure Identification With Greedy Search,” Journal of Machine Learning Research, 2002.

[24] T. van Erven and P. Harremoes, “Rényi Divergence and Kullback-Leibler Divergence,” IEEE Trans. Inf. Theory, vol. 60, no. 7, pp. 3797–3820, 2014.

[25] J. Shlens. Notes on Kullback-Leibler Divergence and Likelihood. 2014

[26] Bongers, Stephan, et al. “Foundations of structural causal models with cycles and latent variables.” The Annals of Statistics, vol. 49, no. 5, 2021

[27] Pearl, Judea. The Causal Foundations of Structural Equation Modeling, 2012

[28] H. A. Simon, “On the definition of the causal relation,” in Models of Discovery, Dordrecht: Springer Netherlands, 1977, pp. 81–92.

[29] K. J. van Stralen, F. W. Dekker, C. Zoccali, and K. J. Jager, “Confounding,” Nephron Clin. Pract., vol. 116, no. 2, pp. c143–c147, 2010.

[30] G. Gigerenzer and W. Gaissmaier, “Heuristic decision making,” Annu. Rev. Psychol., vol. 62, no. 1, pp. 451–482, 2011.

[31] T. A. Stephenson. An introduction to Bayesian network theory and usage. 2000.