The robustness of structures of Bayesian Networks

Wietse van Kooten

TU Delft

Submitted in partial satisfaction of the requirements for the
Degree of Bachelor
in applied mathematics

Supervisor dr. ir. Gabriela Florentina (Tina) Nane

2021-07-09

Preface

As a student of the Delft University of Technology, I single-handedly wrote this thesis in order to finish the bachelor of applied mathematics. Of course, I had a lot of help from many people around me and I would like to thank them all for being there. I would especially like to thank my supervisor dr. Gabriela Florentina (Tina) Nane. Without her, I would not have the knowledge the write this report and I am very grateful for that. I would like to thank dr. Bart van den Dries and Aliza Glasbergen-Plas for providing me with the right tools and techniques to give a presentation and for their support during the preliminary course. I would like to thank all the professors from past years who thought me the fundamental basics of mathematics and who made it possible to write this thesis. Many inspired me with their vision of mathematics. Also, I would like to thank my family for their unconditional support throughout the whole process. I am very grateful.

This thesis will introduce the basic concepts of Bayesian networks. The focus will lay on the learning algorithms and how well they perform. We will try to achieve perfection in search for the best performing learning algorithm and we will even see that data generated from a certain network, may not even be the best fitting Bayesian network for that data. This will immediately raise the question of whether we want to fit the data or predict new observations.

In this thesis, I assume the knowledge learned during the bachelor of mathematics. Nevertheless, I tried to be as clear as possible in every step made. If some concepts still seem vague or new I gladly refer to the books I also used to write this thesis (Scutari and Denis, 2014)(Pearl, 1988). It was a wonderful experience to write this thesis and I hope the reader will also enjoy it.

Abstract

Bayesian Networks are directed acyclic graphs with a lot of information hidden within the graph. There is already done a lot of research on these graphs and they are already applied in many instances. They come back in multiple fields like Health care, finance, psychology and many other fields. The reason for this increasing popularity is the clear visual output and the relatively simple idea of how to apply it. Although the graphical representation can be deceivingly straightforward, the math behind it is not.

A directed acyclic graph that consists of directed arcs and variables in the form of nodes. These arcs represent a certain dependency between the nodes. At the start, I will give a relative small Bayesian network, that only depends on discrete variables. With the use of this example, I will introduce common definitions of a Bayesian network and we will see how the dependencies between nodes are described.

Afterwards, we will look at the process of learning a Bayesian network for which multiple questions are to be considered; Do we already know which variables have a certain dependency? which values can the variables obtain and how much data do we actually have to use for the learning? With multiple learning algorithms, which are based on certain scores or tests, we can construct the structure and the dependencies. When the structure is learned we can focus on the dependencies, the parameters.

The question that will come immediately to mind is how do we know that the constructed Bayesian network is a good one? While that is a complicated question because some of the algorithms that are used to construct the Bayesian networks are based on the scores that give an indication of the goodness of the network. So in this paper, we will create multiple Bayesian networks and take samples of that network. We will apply the learning algorithms on the acquired observations and see how well

they perform. Of course, it is not our only goal to create a Bayesian network that represents the current data in the best possible way. We also want a Bayesian Network which can give a good prediction and it could be that certain structures are better in predicting, while others are better fitted to the data. Besides prediction and goodness of fit we also inspect the robustness of the Bayesian networks. When achieved a solid understanding of these discrete networks we can look at continuous Bayesian networks. These Bayesian networks will in the first instance only consists of normally distributed variables. Although there are a lot of similarities with the discrete Bayesian networks some aspects will be different like the tests and scores. Again, we will be testing the performance of our created structures and will point out the differences. During this analysis we will see how robust the multiple learned structures are.

With the use of R, a language specialised in statistical computing and graphic, we will apply the learning algorithms and use other functions to give us a better understanding of the created Bayesian networks. With the use of Netica and UNINET, we will create our own Bayesian networks and take samples from those Bayesian networks. These samples can be imported to R. We will see how much robustness there is when considering a single arc or even the whole structure from the learning algorithms. We compare the different algorithms and look at the learned structures they came up with and how they differ.

An important question is: When is there a dependency between two variables? Is it in mutual directions and how much certainty do we hope to achieve when creating a network or how much uncertainty are we able to bear? In the end, we hope to have a better understanding of what these Bayesian network show and what they do not show. To know if the learning algorithm is robust for different number of observations. Because the deceivingly robust visual output could actually be misleadingly frail.

Table of Contents

Chapter 1

Introduction

Many before us have already done research to this topic and a lot of people have already applied Bayesian networks in a lot of different fields. The popularity is increasing, but the basic concepts are sometimes forgotten. Learning algorithms are too easily applied without enough understanding and learned structures are taken for granted and other possible structures are neglected. The outcomes can be deceiving and may not be good representations.

This said, Bayesian networks should be used in many instances, because they are great in visualising data and with the knowledge there is, it would be a waste not to use it. Therefore we will try to find more robustness in the learning concept and show what mathematical concepts are grounded in these Bayesian networks.

My goal is to show with use of experiments how frail certain structures are and how a single new observation could possibly affect the whole learned structure. To show how many observations are actually needed to have a good Bayesian network and if we can achieve such a structure, which is robust for any new observation. To have a good Bayesian network we do need to know what our goal is, do we want a Bayesian network that is well fitted to the data or do we prefer a Bayesian network with strong predicting capabilities.

The robustness of Bayesian networks will be found by first considering other aspects. In chapter 2 we will introduce all aspects of graphical models and look at what kind of attributes a Bayesian network has. We will introduce discrete data by means of an example, to show how the structure and dependencies can be viewed. We will

look at how we can use data to learn the structure and the parameters and show that are a lot of algorithms that can find theses structures and parameters. All these learning algorithms are based on test and scores, which we will introduce and show that there are multiple possible scores and tests. This will even further increase the possibilities and we learn a Bayesian network. Therefore we will apply multiple learning algorithms to our example and see how well they perform. In chapter 3, we will introduce Gaussian Bayesian networks that are based on normally distributed variables. In first instance, Bayesian networks based on continuous variables have a lot in common with Bayesian network based on discrete variables, but there are some significant differences. We will look at the differences and setup new scores and tests and see how to apply our learning algorithms. To see if they perform well we apply them on example, which will introduce at the beginning of chapter 3.

In chapter 4, we will bring all our knowledge together and perform multiple tests on multiple Bayesian networks and Gaussian Bayesian networks. We will see which structures are robust and which are not. We can use this information to see how well the multiple learning algorithms perform.

Chapter 2

Discrete Bayesian networks

A good start would be a relatively simple Bayesian network and that is exactly what we are going to do. The first networks we are going to explore will only contain discrete variables, so we are not considering continuous variables. In later chapters, we will discuss the continuous Bayesian networks. Nevertheless, these less complex discrete networks contain a lot of information and could be very insightful. In this chapter, we will look at multiple attributes of Bayesian networks and they will form the building blocks for more complex systems.

2.1 Graphs

Let us first look at graphs in general. In figure 2.1, we see a graph containing two nodes and one directed arc. The nodes represent a variable, which is described within the node, so we see the left node representing the sex of a person and the right node representing education. The arc shows the dependencies between the variables, so in our case, we notice that education is dependent on sex. Later on, we will give a more formal definition of dependency and what it implies.

An important note; the arc gives an intuitive idea that there is a causation. Like

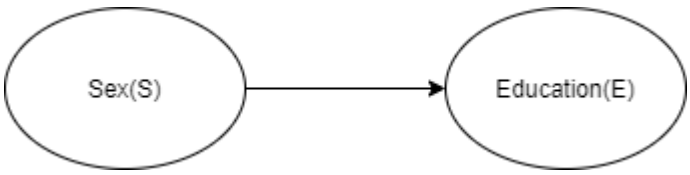


Figure 2.1: A graph containing two nodes and one directed arc

in the graph 2.1 you would think that sex causes the level of education. This does not hold for Bayesian networks. Bayesian notworks speak of a dependence relation instead of a causal relation. There are different models that do consider causal relations.

In this graph 2.1, there is only one arc, which is directed. When a graph only contains directed arcs it is called a *directed graph*. Furthermore, there is a distinction between graphs that are cyclic and graphs that are not. Cyclic graphs contain paths in which it is possible to return to the same node. A path is a kind of route, which is possible by following the directed arcs. In this case, there is only one possible path and that path is not cyclic, so we are looking at a *directed acyclic graph*. A Bayesian network is a directed acyclic graph. Likewise, with dependency, we will state a more formal definition later on.

Graphs can contain a lot of variables and therefore even more arcs because between every variable there could be a certain dependency. This large amount of possible arcs can make a graph overwhelming when considering a lot of variables. So especially in this thesis, we will only consider problems with a relatively small amount of variables.

2.1.1 Introduction of the "Travel use survey" Bayesian network

Now let us introduce a small example of a Bayesian network. In figure 2.2 we see a hypothetical investigation of the usage pattern of different means of transport (Scutari and Denis, 2014). This network contains multiple discrete variables and dependencies. We already stated that it is a Bayesian network, but how do we know it even is a directed acyclic graph? Well, we can immediately note that there are only directed arcs, so we know it is a directed graph. Besides the graph is in such a way represented that we notice that the arcs only point downward, which will give the suggestion that it is not cyclic. When looking for a path that is cyclic we see it is not possible, so we got a directed acyclic graph.

When considering the node "education" we immediately see it is affected by two nodes; "age" and "sex". These nodes are therefore called the *parents* of the "education"

node. Subsequently, what would a parent be without a child? So the "education" node is called the *child* node of both the "age" and "sex" nodes.

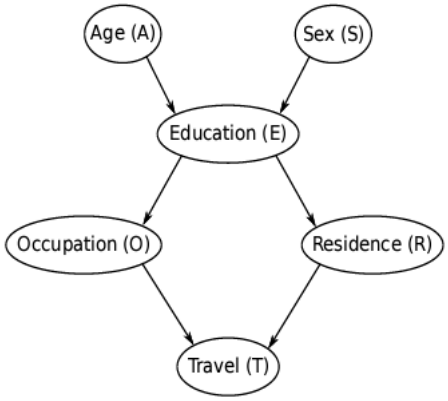


Figure 2.2: A directed acyclic graph

Within a network, we can consider *direct* relations and *indirect* relations. The direct relations of a certain node are all the nodes that are a parent or a child. Indirect relations are nodes for which there is a possible path between the nodes but are not a parent or a child. Think of a great-grandchild or a grandparent. These terms are used frequently when investigating a Bayesian network.

In the Travel use survey, we see that the arcs represent a certain dependency between variables, so we know there are interactions between variables. All the variables are discrete variables, so can only take a countable amount of different values and in our situation, it will only be a few. Let us look at the variables more closely:

1. Age (A); represents the age of a certain individual. This distribution is discretized in the following groups: Younger than 30 is called *young*, between the 30 and 60 is called *adult*, and people older than 60 are called *old*.

2. Sex (S); represents the current sex of an individual. For the simplicity of the example, we will only consider *male (M)* and *female (F)* as an option.

3. Education (E); represents the highest level of education that is completed by an individual. The available levels are: completed *high school (high)* or completed *university (uni)*.

4. Occupation (O); represents the current work situation of an individual. This could be either *employee (emp)* or *self-employed (self)*.

5. Residence (R); this represents the size of the city in which an individual currently lives. This could either be *small (small)* or *big (big)*

6. Travel (T); represents the means of transport of a certain individual. We only considered *car (car)*, *train (train)* and *other (other)*.

Notice that we immediately stated an abbreviation of the possible values a certain variable can take. This will make life easier when considering conditional probability and probability tables.

2.2 Probability

Our variables are discrete, so the joint probability distribution will take a multinomial distribution, which is a generalization of the Bernoulli distribution. In a Bernoulli probability, a variable can only be equal to two different values, like one and zero, or yes and no. In such a way that we can assign a single value for p, for which we can calculate that a certain situation will or will not happen. For a multinomial distribution, it is possible for the variables to take more than two different values, like in our situation. This probability over all the variables is called the *joint distribution*. It is quite cumbersome to write out all the possible combinations for all the different variables, so we will break it down into *marginal distributions*. For each variable, we will notate the marginal distribution.

To obtain these marginal distributions we apply the definition of *conditional probability* and the theorem of *Bayes*.

Definition 2.2.1 (conditional probability) *For $X = (X_1, X_2, \ldots, X_n)$, a set of probability spaces with the unconditional probability of X_i being greater than zero for $i = 1, \ldots, n$, we have:*

$$P(X) = \prod_{i=1}^{n} P(X_i | X_1, X_2, \ldots, X_n) \tag{2.1}$$

Theorem 1 (Bayes) *For X_1 and X_2 two probability spaces with the unconditional probabilities of X_1 and X_2 being greater than zero, we have:*

$$P(X_1|X_2) = \frac{P(X_1 \cap X_2)}{X_2} \tag{2.2}$$

With this definition and theorem, we can split up the joint distribution into multiple marginal distributions, which will be more convenient to work with. The joint distribution over all the variables 2.3 will be quite a puzzle. The multiple variables and possible values they could take, will cause that the complexity of the problem will drastically increase.

If we take a look at the "Travel use survey" example in figure 2.2 we see that the Bayesian network consists of 6 variables for which the joint distribution over all the variables will have the following notation:

$$P(A, S, E, O, R, T) \tag{2.3}$$

Suppose that the nodes that are not connected directly are *conditional independent*.

Theorem 2 (conditional independence) *Suppose variable X_1 and X_2 are conditional independent of another node X_3, then*

$$P(X_1 \cap X_2|X_3) = P(X_1|X_3)P(X_2|X_3) \tag{2.4}$$

Note that for regular independence we have

$$P(X_1|X_2) = P(X_1) \tag{2.5}$$

Of course, this holds for both variables. Take a look at theorem 1, if we apply this to equation 2.4 we see.

$$P(X_2|X_1) = \frac{P(X_1|X_2)P(X_2)}{P(X1)} = \frac{P(X_1)P(X_2)}{P(X_1)} = P(X_2) \tag{2.6}$$

Now if we assume the conditional independence with the non-parental nodes we can

write the joint distribution of the Bayesian network (2.7) as marginals and conditional distributions.

$$\begin{aligned} P(A,S,E,O,R,T) &= \prod_{v\in\{A,S,E,O,R,T\}} P(v|A,S,E,O,R,T) \\ &= P(A)\ P(S)\ P(E|A,S)\ P(O|E)\ P(R|E)\ P(T|O,R) \end{aligned} \tag{2.7}$$

And this will be a lot easier to calculate, but therefore we need to assume independence for indirect connections. We will now look at how we could state that independence.

2.2.1 Graphical and probabilistic independence

The assumption of independence with non-parent nodes comes with some restrictions for the Bayesian network. The question is how can we construct such a network where *graphical separation*($\perp\!\!\!\perp_G$) implies *probabilistic independence*($\perp\!\!\!\perp_P$). With graphical separation, we mean visual independence when looking at a graph. So is there a path between the two nodes or not. Probabilistic independence states the independence between the variables, so are the certain variables independent of each other. We can describe a certain map between these two forms of dependencies. Which represents the "link" between indecencies in distributions and independencies in graphs. First, we will give a more clear explanation of probabilistic independence and graphical separation.

Suppose we have three variables A, E and O of the graph in 2.2, then we know that $O \not\perp\!\!\!\perp_G E$, because we see in the graph that there is an arrow from E to O. Furthermore the same holds for $O \not\perp\!\!\!\perp_G A$, because there is an indirect path. But how can we consider the dependence between O and A if E is already known? This brings us to a new concept.

Definition 2.2.2 (d-separate) *Suppose we have a directed acyclic graph G and we take three disjoint subsets of nodes A, B and C. These disjoint subsets can consist of multiple variables. C is said to d-seperate A from B (is notated as $A \perp\!\!\!\perp B|C$), if along every path between any node in A and any node in B there is a node v satisfying:*

1. *v has converging arcs and neither v nor any of its descendants are in C. Converges will be defined in 2.2.5.*

2. *v is in C and does not have converging arcs.*

So we find by the definition that the graphical separation is clear, such that $O \perp\!\!\!\perp_G A|E$, but will this implicate that it will also hold for the probability. Now let us consider the map between the graphical separation and the probabilistic independence where the implication is considered to be true, so

$$O \perp\!\!\!\perp_G A|E \Longrightarrow O \perp\!\!\!\perp_P A|E \tag{2.8}$$

This implication 2.8 is known as an *independence map*. We can achieve this map if we assume *the local Markov property*.

Definition 2.2.3 (local Markov property) *Each node X_i is conditionally independent of its non-descendants given its parents.*

In this thesis, we will assume that all the maps are at least an independence map. With this assumption, we can obtain equation 2.7. We will look at a brief example of an independence map to get a better idea of the implication.
Suppose we have two networks with the directed acyclic graph of figure 2.1($= G$) and figure 2.3($= G'$). We will immediately notice that the only difference between G and G' is the arc between both nodes. Suppose we got a distribution P with the independence $S \perp\!\!\!\perp_P E$. In this situation figure G' is an independence map because S and E are independent in both figure G' and in P, but figure G is not an independence map of the distribution, because it does not show independence. Strangely enough, both maps are an independence map if the given distribution P_2 is a distribution for which S and E are dependent ($S \not\perp\!\!\!\perp_{P_2} E$) because the independence in the given distribution is empty, which is included in both figures. So sometimes this could be quite misleading and counter intuitive. Let us prove that we can only assume the independence map (Sucheta Nadkarni, 1998). We will prove this with use of *contradiction*.

Figure 2.3: A graph consisting of two nodes and no arc

Suppose we make another assumption. An assumption that our graph is also a dependence map, which states

$$O \perp\!\!\!\perp_P A|E \Longrightarrow O \perp\!\!\!\perp_G A|E \tag{2.9}$$

This will make our Bayesian networks more intuitive. Combining these implications will give a one-to-one relation.

$$O \perp\!\!\!\perp_P A|E \Longleftrightarrow O \perp\!\!\!\perp_G A|E \tag{2.10}$$

This one-to-one relation is called *faithful*. The reason why we can not assume faithfulness is the following. If we look at figure 2.1 we can state the following about the joint distribution:

$$\begin{aligned} P(S,E) &= P(S)\ P(E|S), \quad \textit{which represents the graph in } 2.1 \\ &= P(S)\frac{P(E,S)}{P(S)} \\ &= P(E,S) \\ &= P(E)P(E|S), \end{aligned} \tag{2.11}$$

This implies that both nodes are conditionally dependent on each other and that it is possible to switch the direction of the arc. Because we assumed faithfulness we have property 2.9, so we must draw both arrows in the graph. This will make the graph cyclic, so our graph will not be a Bayesian network anymore, which is a contradiction. Therefore we do not assume a dependence map.

If we just have one of the two arcs in 2.1, then it still is an independence map. It

will not be faithful, because probabilistic independence no longer implies graphical independence. This because we see

$$\begin{aligned} &S \not\perp\!\!\!\perp_G E \\ &S \perp\!\!\!\perp_P E, \ so \\ &S \perp\!\!\!\perp_P E \not\Longrightarrow S \perp\!\!\!\perp_G E \end{aligned} \tag{2.12}$$

Now that we have the assumptions for a Bayesian network we can formally define a Bayesian network.

Definition 2.2.4 *Let G be a directed acyclic graph with $a \in A$ the directed arcs that in the graph. Given a probability distribution P on a set of variables X, directed acyclic graphs $G = (X, A)$ is called a Bayesian network and denoted $B = (G, X)$ if and only if G is a minimal indepence map of P , so that none of its arcs can be removed without destroying its independence mapness.*

This definition will be our hunting license to create some beautiful Bayesian networks.

Do notice that if we had considered the whole graph 2.2, we would have had a faithful map. This is because in that situation it is not possible to reverse an arc. This raises the question of when and when it is not possible to reverse an arc.

Equivalence classes

We just saw in the graph 2.1 and equation 2.11 that it is possible to reverse an arc and that the new graph is still equivalent to the former graph. This will raise the question if there are multiple equivalent graphs for a certain directed acyclic graph. Let us first consider three types of structures that can be found in a Bayesian network (Scutari and Denis, 2014).

Definition 2.2.5 (fundamental connections) *For a directed acyclic graph there are 3 types of connections between 3 variables. For sake of simplicity, we will consider figure 2.2.*

- *Serial connection; consider the nodes A, E and O. We can simplify this connection as followed:*

$$A \rightarrow E \rightarrow O$$

  This form of connection is called serial since both arcs have the same direction.

- *Divergent connection; consider the nodes R, E and O. We can simplify this connection as followed:*

$$R \leftarrow E \rightarrow O$$

  This form of connection is called divergent since both arcs diverge from each other.

- *Convergent connection; consider the nodes A, E and S. We can simplify this connection as followed:*

$$A \rightarrow E \leftarrow S$$

  This form of connection is called convergent since both arcs converge to E.

A convergent connection is also called a *v-structure*. These v-structures are very important, because:

1. If a node forms a v-structure with other nodes, then their arcs can not be reversed.

2. If by reversing an arc there will arise a new v-structure it is not possible

This will be our rule of thumb when considering equivalent directed acyclic graphs. So when we consider the example "travel use survey" with the graph 2.2 we see that arcs $A \rightarrow E \leftarrow S$ and $O \rightarrow T \leftarrow R$ already form a v-structure, so cannot be reversed. For $E \rightarrow O$ and $E \rightarrow O$ the second condition holds because if we change either one of them we will create a new v-structure. So in this case the graph is also a faithful map. This immediately implies that there is not an equivalent form.

For graphs in which we can reverse certain arcs, it is convenient to describe the graphs in such a way that we capture all the equivalent forms in one. Which will bring us to a new concept.

Definition 2.2.6 (Markov blankets) *Suppose we have a set of random variables* $S = X_1, \ldots, X_n$ *and we choose a certain* $Y \in S$ *and we got a subset* $S_1 \subset S$ *. The markov blanket* Y *is equal to* S_1 *for which holds*

$$Y \perp\!\!\!\perp_G (S \setminus S_1) \mid S_1 \tag{2.13}$$

Which in the first instance seems a little vague, but the definition states that a Markov blanket consists of:

- All parent nodes
- All children nodes
- All nodes with which it forms a v-structure

With the use of these Markov blankets, we will be able to construct certain graphs which include all the equivalent forms. These graphs are called *moral graphs*. Moral graphs are undirected graphs with the following properties:

- Each v-structure is shown with direction, so these are equivalent as in the directed acyclic graph
- Each arc which will create a v-structure if reversed will stay a directed arc
- All other arcs will become an undirected arc; so the arcs can be reversed.

This means that the moral graph of the "travel use survey" will be exactly the same as the directed acyclic graph, which implies there are no equivalent graphs. With the use of these moral graphs, we can easily inspect if there are any equivalent graphs because for any undirected arc we can give any direction and we will obtain multiple equivalent graphs.

Now we achieved enough knowledge about the structure of the Bayesian network that we can focus on the next part; the parameters of a Bayesian network.

2.2.2 Probability tables

In 2.7 we saw that the joint distribution over all variables can be written down in multiple marginal and conditional distributions. These distributions are easier to work with. Let us look at the Bayesian network given in figure 2.2. Suppose we would have written the joint distribution down, then we would need to have a parameter for every possible value of each variable. That would make 144 unique parameters for our small network. This number will increase exponentially when either the variables increase or the possible values of the variables. In the case of writing it down in multiple marginal and conditional distributions, there will only be 21 parameters required. Besides, the math will be less cumbersome.
So for each node, we will make a table where will notate the possible values of our node and the possible values of its parents. For each combination, there will be a unique value representing the probability of occurring. So suppose we got the following *probability table* 2.1.

Travel	car	train	other
small & emp	0.48	0.42	0.10
small & self	0.56	0.36	0.08
big & emp	0.58	0.24	0.18
big & self	0.70	0.21	0.09

Table 2.1: Probability table of the variable Travel

In the table, we see the conditional probability of the means of travel. It becomes clear that the means of travel depends on the occupation and the residence. As long as the values of occupation and residence are known it will not matter what values the other variables take. In the case that those values are not known, other nodes could possibly influence the node. With the knowledge of how the parameters and the structure looks, we can try to learn our own networks.

2.3 learning

Of course, we do not always have the privilege of a known structure or a probability table, which depends on the structure. Therefore we need to be able to construct

both with the use of the given data. When learning the probability tables, which is referred to as *parameter learning*, we will assume that the structure is known and that the given Bayesian Network is a well-defined one, so according to the definition 2.2.4. When there is not a known structure for the Bayesian network we will construct one with the use of *structure learning*. There are multiple learning algorithms to construct the structure. For this, we do need enough data and when there is no clear dependency it will be harder to learn the structure. So later on we will look at how the algorithms will perform on the given data and how they differ from each other. With multiple scores, we will try to get a better understanding of which given network is the best. Let us first look at the learning concept in a more general and abstract way. Suppose we are given a dataset, called D, and we want to determine the structure of a Bayesian network, which we will call B. Because we want to calculate a Bayesian network which is a well-defined one, like the one we proposed in 2.2.4, it will consist of a DAG called G and a set of variables X. Let us then introduce a new variable Θ, which represents the parameters of the joint distribution of X. Then it is possible to assume without loss of generality that Θ uniquely identifies B. In other words, when we apply structure learning we investigate G with the condition that D is known. When we have found the optimal G, the graph with the structure, we use it to find Θ, so we learn the parameters conditioned that both G and D are known.

$$P(B|D) = P(G, \Theta|D) = P(G|D) \cdot P(\Theta|G, D) \tag{2.14}$$

Equation 2.14 perfectly reflects the two parts of the problem we try to investigate. The structure learning part ($= P(G|D)$) and the parameter learning part ($= P(\Theta|G, D)$) (Scutari and Denis, 2014).

2.3.1 Structure learning

Before we dive into the multiple algorithms of learning the structure of a Bayesian network, it is important to know that we try to model the given data as best as possible. Because we try to approach reality as close as possible, we should listen to experts of that specific expertise, also known as *expert knowledge*. They can

pinpoint certain dependencies, which gives us a better understanding of how the learning algorithm should construct the Bayesian network. In the algorithms, we will introduce possibilities to state beforehand which arcs are favourable, *white lists*, and even which arcs are unwanted or even impossible in reality, *blacklists*. In this thesis we will not make use of expert knowledge, so the learning process is only based on the data.

There two main ways of learning a structure. The first possibility is the look at an arc and investigates the independence between the two nodes and whether or not that arc should be present. These classes of statistical criteria are called *conditional independence tests*, which are used in *constraint-based* algorithms. And *network scores*, which are used in *score-based* algorithms. These scores are a goodness-of-fit statistics measuring how well the DAG mirrors the dependence structure of the data. We will look at algorithms that are constraint-based, so based on tests, and algorithms based on scores.

Another important note is that we need to make some assumptions before we can apply any structure learning. These assumptions may seem somewhat obvious, but nevertheless still important and are forgotten sometimes.

- There is a one-to-one relationship between the nodes and the variables in X. So it is not possible that a certain variable is represented in multiple deterministic functions (Nodes).
- The only relationship which exists within the Bayesian network are dependence relations, so independencies, dependencies and conditional independencies.
- The observations are independent of each other.
- Every combination of values the variables can obtain must represent a valid event (Pearl, 1988).

Network scores

Looking back at equation 2.14 we can further decompose $P(G|D)$, which represents the structure learning which we want to maximise and is also known as the *posterior*

distribution. The decomposition is done with the use of theorem 1 (Bayes).

$$\begin{aligned} P(G|D) &= \frac{P(D|G)P(G)}{P(D)} \\ &\propto P(D|G)P(G) \\ &= P(G)P(D|G,\Theta)P(\Theta|G) \end{aligned} \tag{2.15}$$

This decomposition helps us to divide the structure learning into two parts. $P(G)$ is called the *prior information* and as mentioned above, we sometimes already got some expert knowledge about the DAG. Within the prior information, we can process this information. The second part, $P(D|G)$, is the marginal likelihood of the data given G averaged over all possible parameter sets Θ.

The problem now is that we do not want that the structure learning depends on the parameters. That would be quite inconvenient because the parameters are based on the structure. Therefore we will integrate Θ out of the equation.

$$P(G)P(D|G,\Theta)P(\Theta,G) = P(G)\int P(D|G,\Theta)P(\Theta|G)\,d\Theta \tag{2.16}$$

Suppose we have a certain Bayesian network with n variables. We can further decompose 2.16 to the marginal distributions and conditional, as mentioned in 2.1 and with the assumption that a variable only depends on its parents we will get. We will immediately introduce a new notation; Π_{X_i} are the parents of the node X_i.

$$\begin{aligned} P(D|G) &= \int \prod_{i=1}^{n} [P(X_i|\Pi_{X_i},\Theta_{X_i})P(\Theta_{X_i}|\Pi_{X_i})d\Theta] \\ &= \prod_{i=1}^{n} [\int P(X_i|\Pi_{X_i},\Theta_{X_i})P(\Theta_{X_i}|\Pi_{X_i})d\Theta_{X_i}] \\ &= \prod_{i=1}^{n} E_{\Theta_{X_i}}[P(X_i|\Pi_{X_i})] \end{aligned} \tag{2.17}$$

$P(D|G)$ can be computed in a reasonable time if we assume multinomial distribution in our discrete Bayesian network. We assume

$$X_i|\Pi_{X_i} \sim \ multinomial(\Theta_{X_i}|\Pi_{X_i}) \tag{2.18}$$

We will make use of the *Dirichlet distribution*.

Definition 2.3.1 (Dirichlet distribution) *the Dirichlet distribution is a multivariate generalisation of the beta distribution. So a Dirichlet distribution with two components is a beta distribution (with the summation over the components equal to 1). It is often denoted with* $Dir(\boldsymbol{a})$, $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$. *With the following density function.*

$$\begin{aligned} P(X = x) =& \frac{\Gamma(\sum_i a_i)}{\Gamma(a_1)\Gamma(a_2)\ldots\Gamma(a_n)} x_1^{a_1} x_2^{a_2} \ldots x_n^{a_n} \\ & \textit{with the assumption}: \sum_i x_i = 1,\ x_i \in [0,1],\ a_1, \ldots, a_n > 0 \end{aligned} \tag{2.19}$$

The Dirichlet distribution is a conjugate prior for the multinomial distribution. This means that if the prior distribution of the multinomial parameters is Dirichlet then the posterior distribution is also a Dirichlet distribution (with parameters different from those of the prior). The benefit of this is that

- the posterior distribution is easy to compute
- it is in some sense possible to quantify how much our beliefs have changed after collecting the data.[1]

For the special case of the multinomial distribution, so in our case 2.18, we have $(\Theta_{X_1}, \ldots, \Theta_{X_n})$ be the vector of multinomial parameters. If

$$(\Theta_{X_1}, \ldots, \Theta_{X_n}) \sim Dir(a_1, \ldots, a_n) \tag{2.20}$$

Suppose we condition on $(\Pi_{X_1}, \ldots, \Pi_{X_n})$, then we will get

$$(\Theta_{X_1}, \ldots, \Theta_{X_n} | \Pi_{X_1}, \ldots, \Pi_{X_n}) \sim Dir(a_1 + x_1, \ldots, a_n + x_n) \tag{2.21}$$

The uniform distribution is actually a special case of the Dirichlet distribution, cor-

[1] Found this on an online forum where people can pose questions. This forum is known as Stackexchange, `https://stats.stackexchange.com/questions/44494/why-is-the-dirichlet-distribution-the-prior-for-the-multinomial-distribution`

responding to the case $a_1 = a_2 = \ldots = a_n = 1$.The fact that the Dirichlet class includes these natural *non-informative* priors is another reason for using it. We will use the posterior Dirichlet 2.21 to solve $P(D|G)$, which is known as *Bayesian Dirichlet equivalent uniform* (BDeu) score (David Heckerman and Chickering, 1995). Since it is the only score used within the BDe family it is commonly called BDe. BDe assumes a flat prior over both the space of the DAGs and the parameter space of each node.

$$P(G) \propto 1 \qquad and \qquad P(\Theta_{X_i}|\Pi_{X_i}) = a_{ij} = \frac{a}{|\Theta_{x_i}|} \tag{2.22}$$

Let us notate the closed form expression for 2.17, known as the Bayesian Dirichlet (BDe) score (David Heckerman and Chickering, 1995):

$$\begin{aligned} BDe(G, D) &= \prod_{i=1}^{n} BDe(X_i, \Pi_{X_i}) \\ &= \prod_{i=1}^{n} \prod_{j=1}^{q_i} \left[\frac{\Gamma(a_{ij})}{\Gamma(a_{ij} + x_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(a_{ijk} + x_{ijk})}{\Gamma(a_{ijk})} \right] \end{aligned} \tag{2.23}$$

This holds for multinomial distributions assumed in discrete Bayesian networks. $P(D|G)$, the marginal likelihood, can be estimated with the *Bayesian Dirichlet equivalent uniform*, also called the *BDe*. With:

- n represents the number of nodes in G
- r_i represents the number of categories for the node X_i
- q_i represents the number of configurations of the categories of the parents of X_i
- x_{ijk} represents the number of samples which have the jth category for node X_i and kth configuration for its parents.

Another network score we will look into is the *Bayesian Information criterion*, also known as BIC, which is an approximation of $P(D|G)$.

$$BIC(G, D) \rightarrow log\ BDe(G, D), \qquad as\ the\ sample\ size\ n \rightarrow \infty \tag{2.24}$$

When applying the logarithm function to 2.23 we will get a more simplistic expression that will only depend on the likelihood function.

$$BIC(G, D) = \sum_{i=1}^{p} \Big[logP(X_i|\Pi_{X_i}) - \frac{|\Theta_{X_i}|}{2} log\ n) \Big] \tag{2.25}$$

There are multiple algorithms that will try to maximise these networks scores. They could be divide into three groups: *Greedy searches*, *genetic* and *simulated annealing*. We will only look into algorithms that are greedy searches. When implementing these algorithms in R, later on, we will give a broad overview of how they work and what kind of results they give.

Conditional independence tests

As we mentioned before conditional independence tests are used to look at individual arcs and to state whether or not they should exist. So if we look at such a scenario we can have the following problem. Suppose we got the following variables of the "Train use survey" 2.2 (See table 2.1.1 for the full explanation):

- Sex (S); with the values: *Male (M)* and *female (F)*.
- Education (E); with the values: *High school (high)* or *University (uni)*.
- Residence (R); With the values: *Small (small)* or *big (big)*

And we would like to know if we should add an arc from S to R. Then there are 2 outcomes. One, the probability that there exists such an arc is high enough to add the arc. Or two, the probability that there exists such an arc is negligible that will keep our current network. Because there does exist a path from S to R, through E we assume to already know the outcome of E, otherwise, we will also take that dependency into account. So our hypothesis looks like this.

$$H_0 : R \perp\!\!\!\perp_P S \mid E \tag{2.26}$$

This will be our null hypothesis of independence and our alternative hypothesis look like this.

$$H_1 : R \not\perp\!\!\!\perp_P S \mid E \tag{2.27}$$

with the *mutual information* or the *Pearson's* test we can set up our test. The tests will have the following forms;

Let us first introduce a new variable $\boldsymbol{Z}$ are the group of variables on which we will condition. The *mutual information* test is defined as (Agresti, 2013),

$$MI(X, Y|\mathbf{Z}) = \sum_{i=1}^{R}\sum_{j=1}^{S}\sum_{k=1}^{E}\frac{n_{ijk}}{n}log\frac{n_{ijk}n_{++k}}{n_{i+k}n_{+jk}} \tag{2.28}$$

This is almost equivalent to the *log-likelihood ratio* test G^2, which differs bij a factor of $2n$, with n the sample size. *Pearson's* test is defined as

$$\begin{gathered} X^2(X, Y|\mathbf{Z}) = \sum_{i=1}^{R}\sum_{j=1}^{S}\sum_{k=1}^{E}\frac{(n_{ijk} - m_{ijk})^2}{m_{ijk}} \\ where\ m_{ijk} = \frac{n_{i+k}n_{+jk}}{n_{++k}} \end{gathered} \tag{2.29}$$

So, we got n_{ijk}; the number of observations for the possible combinations of the variables. So in our case there are $2 * 2 * 2 = 8$ different possible combinations, because each variable has two possible values. "+" as subscript implicates a sum over an index, so

$$n_{+jk} = \sum_{i=1}^{R} n_{ijk} \tag{2.30}$$

This null hypothesis of independence can be tested using:

- The asymptotic chi-squared distribution, which requires an adequate sample size to be accurate.
- The Monte Carlo permutation. This test is unbiased and works good for a few observations (Edwards, 2000).
- The semiparametric permutation, which is a compromise between the two other tests (Tsamardinos and Borboudakis, 2010).

For the sake of simplicity, we will focus only on the asymptotic chi-squared distribution under the null hypothesis. the chi-square distribution with k degrees of freedom is the distribution of a sum of the squares of k independent standard normal random variables. In our "travel use survey" example we got a summation over 8 different

possible combinations, so we will get a chi-squared distribution with $k = 8$. The null hypothesis is rejected for large values of the test statistic. This value will increase with the strength of the conditional dependence between the nodes R and E. How higher the value, how more indication there is to reject the null hypothesis.

2.3.2 Parameter learning

When the structure is learned we can further work out our Bayesian network by finding the parameters. There are two customs ways to do this.

Maximum likelihood estimates

One way to approach this learning algorithm is with the *maximum likelihood estimates*. In a discrete situation, we can optimise each empirical frequencies to see which dependency between two arcs is the strongest. So for each possible combination of variables and for each possible value, we can calculate the frequency, which will give us the following for a single case.

$$\begin{aligned} \hat{P}(Travel = car | Sex = male) &= \frac{\hat{P}(Travel = car, Sex = male)}{\hat{P}(Sex = male)} \\ &= \frac{\#Travel = car \ and \ Sex = male}{\#Sex = male} \end{aligned} \tag{2.31}$$

This is also known as the *frequentist* method.

Bayes estimates

We can also apply the Bayesian approach, so we will view the parameters as random variables; the estimated posterior probabilities are computed from a uniform prior over every condition probability table (Scutari and Denis, 2014). In other words, we will make use of a non-informative prior, so our learned conditional tables will be based on the posterior. Like in the Dirichlet posterior we got an imaginary sample size a_i, which indicates how strong the influence is of the prior distribution compared to the posterior (based on the data). This can be viewed as an imaginary sample

supporting the prior. The value of a is divided by the number of cells in the conditional probability table and used to compute the posterior estimate as a weighted mean with the empirical frequencies. Suppose we apply this to our "Train use survey" example and we take a sample size of n and we look again at the variables T and S. We take the likelihoods:

$$\begin{aligned} \hat{p}_{car,male} &= \frac{\#Travel = car \ and \ Sex = male}{n} \\ \hat{p}_{male} &= \frac{\#Sex = male}{n} \end{aligned} \tag{2.32}$$

And the priors:

$$\pi_{car,male} = \frac{1}{T_{levels} * E_{levels}} \qquad , \qquad \pi_{car} = \frac{T_{levels}}{T_{levels} * E_{levels}} \tag{2.33}$$

With the levels of a variable equal to the number of different values, it can obtain. This will give us

$$\begin{aligned} \hat{P}(T = car, Sex = male) &= \frac{a}{n+a}\pi_{car,male} + \frac{n}{n+a}\hat{p}_{Travel=car,Sex=male} \\ \hat{P}(Sex = male) &= \frac{a}{n+a}\pi_{male} + \frac{n}{n+a}\hat{p}_{male} \end{aligned} \tag{2.34}$$

Thus

$$\hat{P}(T = car | Sex = male) = \frac{\hat{P}(T = car, Sex = male)}{\hat{P}(Sex = male)} \tag{2.35}$$

As mentioned before, we choose a non-informative prior, so a is very small (often equal to 1). We sometimes increase a for less extreme values, so the probabilities will be closer to a uniform one.

2.4 Applying the learning algorithms

To see how the learning algorithms perform we will apply them to the "Train use survey", see figure 2.2. We modelled the example in Netica, which looks like this 2.4. Netica is a powerful, easy-to-use, complete program for working with, among other things, Bayesian networks. It works quite intuitive and has clear visual output. For discrete Bayesian networks, we will often make use of Netica. To apply the learning algorithms we first need to have data to apply them on. So we will generate our

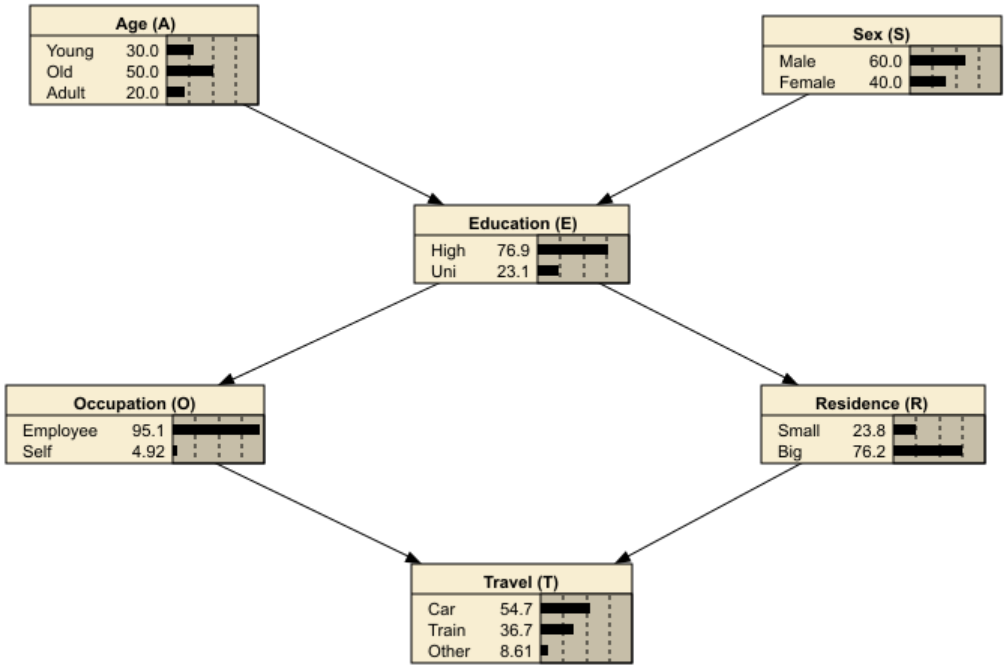


Figure 2.4: Bayesian network of the "Travel use survey". Constructed with use of Netica

samples. Netica has a function to generate samples of the Bayesian network that was implemented. We will generate one million observations and we constantly use different sizes of that sample to see how good the algorithms will perform with different amounts of data points.

2.4.1 Learned structures

We will first show the results of six learning algorithms on the sample and discuss some aspects of the data. Thereafter we will discuss the algorithms in more detail, so do not worry when hearing some algorithms that do not sound familiar. When applying the learning algorithms on the first 1000 observations we got the following results 2.5.

In the top row, we find the two structures learned by the hill-climbing (hc) algorithm on the left based on the BIC score and on the right based on the BDe score. In the second row, we find the two graphs constructed with the use of the Grow-shrink (GS) algorithm on the left based on the mutual information (mi) test and on the right based on the Pearson's test (x2). In the last row, we find the two graphs

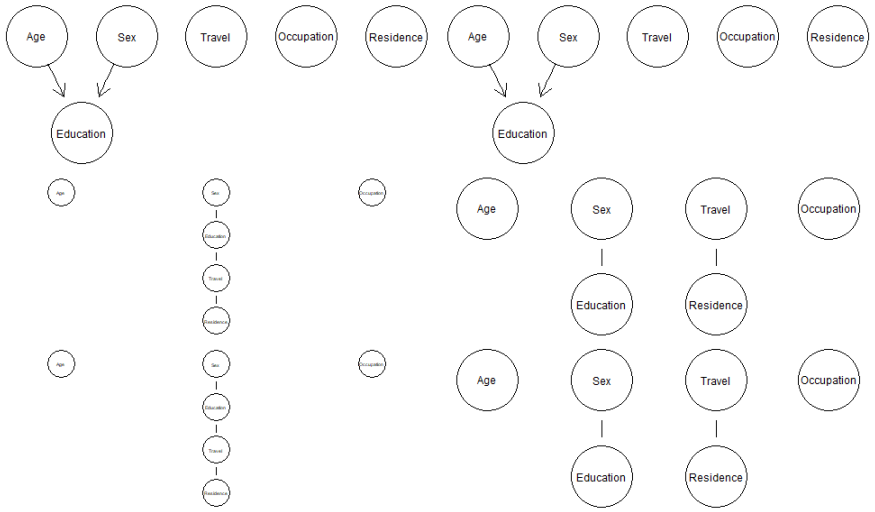


Figure 2.5: Bayesian network of "Travel use survey" with 1000 observations. From left to right we got on the top: hc based on BIC, hc based on BDe. On the middle: GS based on mi, GS based on x2. On the bottom row: iamb based on mi, iamb based on x2

learned by the Incremental association method (iamb) on the left based on the mutual information test (mi) and on the right based on the Pearson's test (x2).

It will be immediately clear that 1000 observations are not enough to create a model for the example. Nevertheless, we do already see some learning and we can see the differences between the learning algorithms. All algorithms show a dependency between Sex and Education. Notice that the learning algorithms GS and iamb create moral graphs. To understand why that specific dependency is most easily learned we will need to look at the probability tables, which look like this.

Education	High	Uni
Young & Male	0.75	0.25
Adult & Male	0.72	0.28
Old & Male	0.88	0.12
Young & Female	0.64	0.36
Adult & Female	0.70	0.30
Old & Female	0.90	0.10

Residence	Small	Big
High	0.25	0.75
Uni	0.2	0.8

Occupation	Employee	Self Employee
High	0.96	0.04
Uni	0.92	0.08

textbfTravel	car	train	other
small & emp	0.48	0.42	0.10
small & self	0.56	0.36	0.08
big & emp	0.58	0.24	0.18
big & self	0.70	0.21	0.09

In the probability tables, we see the multiple dependencies between the variables. When observing the probability tables we can look at the change when we change one of the conditioned variables. So suppose we know that the person is old we immediately see in the probability table that the probability of achieving a university degree is lower. The same holds for the sex of the person, this increases the probability of achieving a university degree. Of course, we only got only 1000 observations so there could be a lot of fluctuation. For example, the probability of only having achieved a high school diploma and being self-employed is 0.04, which is rather small. Besides not all observations consist of people who only achieved a high school diploma, so that certain combination will be rare in our dataset.

Let us view a small summary of the 1000 observations.

Age	Sex	Education	Travel	Occupation	Residence
Adult : 193	Female : 420	High : 755	Car : 568	Employee : 962	Big : 764
Old : 493	Male : 580	Uni : 245	Other: 70	Self employee : 38	Small : 236
Young : 314			Train : 362		

We immediately see only 38 people are self-employed, so it will be harder to learn a dependency. Notice that the generated data is not exactly equal to the probability of a certain variable. This is because it is generated dataset, so there is randomness within the sample. Before we apply the learning algorithms we will take a closer look at how they work.

2.4.2 Hill climbing

One of the learning algorithms we will be applying is a score based function *hill-climbing*. Hill climbing has the following algorithm (Scutari and Denis, 2014).

1. Choose a network structure G over V, which is empty if no *white-list* is added.
2. Compute the score of G, which we be notated as $score_G = score(G)$.
3. Set a new variable equal tot that score $maxscore = score_G$.
4. Repeat the following steps as long as $maxscore$ increases:

- for every possible change; addition, deletion or reversal of an arc which will not result in a cyclic network or an arc in the *blacklist*:

  (a) compute the score of the changed network G^*, $score_{G^*} = Score(G^*)$

  (b) if $score_{G^*} > score_G$, set $G = G^*$ and $score_G = Score_{G^*}$

5. Return G

So in this algorithm, we are searching for a single change, which will decrease the score the most of all the possible modifications, and if that newly acquired network is better than our current network we set our current network equal to the newly acquired network. This will be repeated until there is not a single modification that will decrease the score. Notice that a decrease in the score is a sign that the Bayesian network performs better. Of course, we got two different score functions on which the algorithm is based BDe and BIC, but we already have seen that those will not differ that much from each other.

2.4.3 Grow shrink

The *Grow-shrink(GS)* algorithm is based on finding the Markov blanket (Margaritis, 2003). The GS algorithm uses the concept of a Markov blanket of a variable. We already saw what Markov blankets are. Note that a Markov blanket of a variable T is denoted with MB(T). MB(X) d-separates, see 2.2.2, X from any other variable outside MB(X). In other words, MB(X) contains all the variables in the graph carrying information about X. The GS algorithm focuses on the recovery of MB(X) using pairwise independence tests. It consists of two phases. In the first growing phase, MB(X) is initially an empty set, denoted by S. Then the algorithm adds variables to S as long as they are associated with X given the current contents of S. In this phase, even variables not really belonging to MB(X) could be added to S. The second, shrinking, phase is performed to identify and remove these variables (Flaminia Musella and Vitale, 2019).

2.4.4 Iamb

The last algorithm we will use for the discrete Bayesian networks is *Incremental Association (iamb)*(Ioannis Tsamardinos and Statnikov, 2003). IAMB consists of a forward and a backward phase. First, an estimate of the MB(T) is kept in the newly defined set CMB. In the forward phase, all variables that belong in MB(T) are put in CMB, this could also include *false positives*. False positives are in this case variables that are not really in the markov blanket of T. The backward phase will identify the false positives and remove them, such that CMB = MB(T). The heuristic used in IAMB to identify potential Markov Blanket members in phase I is the following:

- starts with an empty set for the CMB
- add the variable that maximizes a test function $f(\mathrm{X};\ \mathrm{T} \mid \mathrm{CMB})$. f should return a non-zero value for every variable in the Markov Blanket for the algorithm to be sound, and typically it is a measure of association between X and T given CMB. For f we look at two possible choices Pearson's and the mutual information test

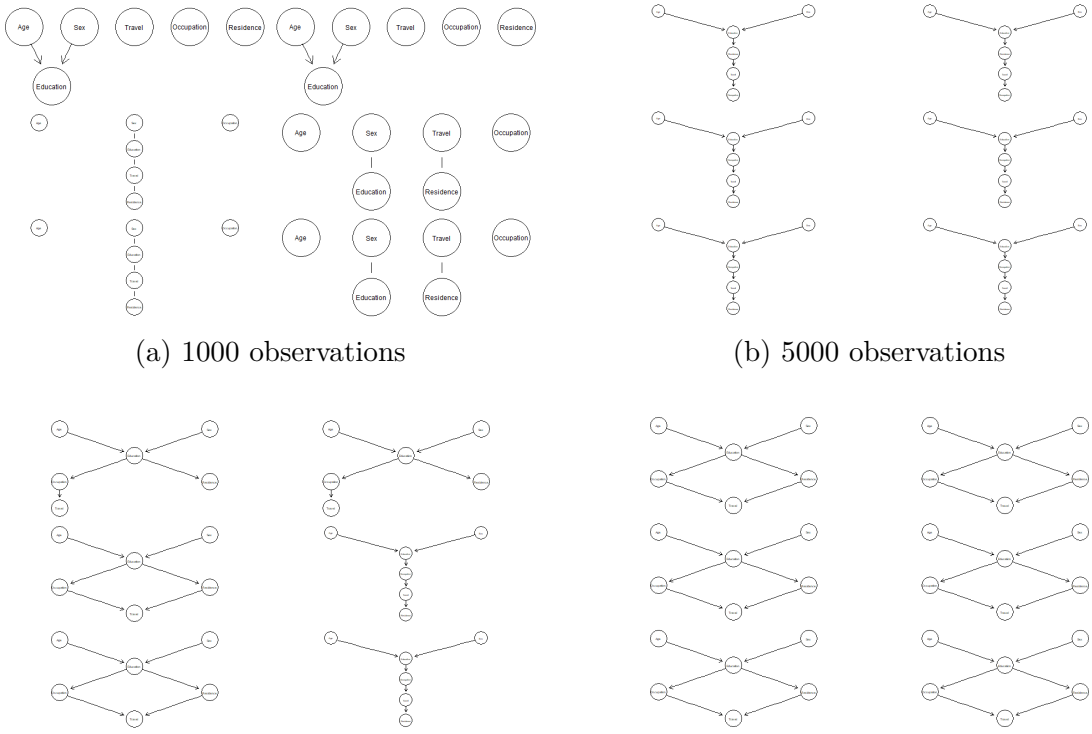
It is important that f is an informative and effective heuristic so that the set of candidate variables after the forward phase is as small as possible. This is convenient for two reasons

1. time efficiency
2. sample efficiency; sample do not have to be larger than necessary to perform the conditional tests of indepedence.

In the backward conditioning, we remove the features that do not belong to the MB(T). This is done by testing whether a feature X from CMB is independent of T given the remaining CMB ($X \perp\!\!\!\perp_P T \mid \mathrm{CMB}$).

2.4.5 Learned structures for more observations

We have seen how the learning algorithms perform with 1000 observations, which was not that good. Let us look at performances with 5000, 10000 and even 15000 observations.



(a) 1000 observations

(b) 5000 observations

(c) 10000 observations

(d) 15000 observations

Figure 2.6: The learned structure for 1000, 5000, 10000 and 15000 observations. For each figure we see from left to right on the top: hc based on BIC, hc based on BDe. In the middle: GS based on mi, GS based on x2. On the bottom row: iamb based on mi, iamb based on x2

After 1000 observations there is some improvement, but still far from what we aim to learn. We do see that after 5000 observations we learned some good-looking Bayesian networks, so notice that we at least need 1000 observations to learn meaningful Bayesian networks. This is quite staggering for the small Bayesian network we are considering. In chapter 4 we will discuss these results more thoroughly and look at other aspects of the structure learning.

2.4.6 Learned parameters

After 15000 observations we found for every structure learning algorithm a structure which is equivalent to the one it is based on. So that is a nice result. Let us now look if we can find equivalent parameters for our structure to complete our learned BN. We will make use of the observations we used to find the true structure. First we will apply maximum likelihood and then Bayes estimator. The code for calculating can be found in A.1

Maximum likelihood estimator

When applying the maximum likelihood estimators on the Travel node we found the results, which can be seen in A.1.1. These results are not really great, because their probabilities are for some parameters almost 0.1 off. That is a lot. Let us look at the Bayes estimators.

Bayes estimator

When applying the Bayes estimator with an imaginary sample size of 10 we find the resutlts which can be seen in A.1.1. The Bayes estimator produces almost the same probabilities. So we can not really state which one is better, but it seems like that for this sample we found some odd probabilities. When applying these estimators for 1 million observations there all within a range of 0.01 of the true value, so that is a lot better. This also indicates that is quiet hard to learn discrete BNs.

Chapter 3

Continuous Bayesian networks

We have gained a lot of knowledge about discrete Bayesian networks and we now acquired a good foundation to explore more Bayesian networks. In this chapter, we will focus on Bayesian networks containing continuous variables. In the first instance, we will only consider normally distributed variables and later on we will change the distributions. We will assume a multivariate Normal (Gaussian). Gaussian Bayesian Networks (GBNs), in which X is multivariate normal and the X_i given $P(X_i)$ are univariate normals defined by the linear regression model. We first look at a small example and during that example, we will introduce multiple aspects of these Bayesian networks.

Throughout this chapter, we will find that there are some differences when considering continuous variables compared to discrete variables. We will look at the difference and what the consequences are. If a certain aspect is not explicitly mentioned, like the equivalence classes, you can assume that it is similar to the discrete Bayesian network.

When considering GBNs we make a few assumptions:

1. Every node follows a normal distribution;
2. Nodes without any parent, known as root nodes, are described by the respective marginal distributions;
3. The conditioning effect of the parent nodes is given by an additive linear term in the mean, and does not affect the variance. In other words, each node has

a variance that is specific to that node and does not depend on the values of the parents;

4. The marginal distribution of each node can be equivalently expressed as a Gaussian linear model which includes an intercept and the node's parents as explanatory variables, without any interaction term.

The fourth item mentions Gaussian linear models, which are important considering GBNs, so we will introduce a definition.

Definition 3.0.1 (Gaussian linear regression model) *Suppose we have the predicter multivariate X with observations $x = (x_1, \ldots, x_p)$ and the dependent variable Y normally distributed (Fetsje Bijma and Vaart, 2016). Then*

$$Y = \sum_{i=1}^{p} \beta_i x_i + e \tag{3.1}$$

With

- *Measurement error e, for which holds: $e = Y - \sum_{i=1}^{p} \beta_i x_i$*
- *Parameter vector θ, for which holds $\theta = (\beta_1, \ldots, \beta_p, \sigma^2)$*
- *The intercept is equal to the expectation of Y if all regression coefficients $\beta_1, \ldots, \beta_p = 0$*

For this definition we make the following assumptions:

- *The expectation of Y depends on x, but the variance does not*
- *The expectation of Y is a linear function of x*
- *The measurement error is normally distributed*

3.1 A new network

As in chapter 2, we will introduce all the new concepts with the use of an example. So let us introduce an example. We will be doing research on the harvested grain mass of a certain plant, a "crop analysis" (Scutari and Denis, 2014). A lot of factors

come into play when we are doing the "crop analysis", but we made life easy and simplified the experiment by defined the following variables.

- *Environmental potential (E)*; think of location, season etc. It is a synthetic variable that represents the initial status of a plant.
- *Genetic potential(G)*; represents the effect of the genotype of the plant. This is also a synthetic variable which the initial status of a plant.
- *Vegetative organs (V)*; the leaves, stems, root etc. It considers the vitality of these organs. This variable is influenced by the two initial variables E and G
- *Number of seeds(N)*; indicates the number of seeds at the flowering time. This variable is dependent on the vegetative organs.
- *Seeds mean weight(W)*; the mean weight of the seeds. This variable is also dependent on the vegetative organs.
- *Crop (C)*; the harvest grain mass, which is dependent on the number of seeds and the mean weight.

We also made this example in UNINET (see figure 3.1), which is great for creating Bayesian networks with continuous data. UNINET is software developed at DIAM, TU Delft, Bayesian networks to model non-parametric Bayesian networks. With the use of UNINET we are also able to create samples that can be imported into R to do our analysis.

We immediately see a difference with regard to the discrete Bayesian networks. We do not consider probability tables anymore, instead, we have another way of expressing the dependency. Do note that this example is quite similar to the one we saw in chapter two (directed acyclic graph 2.2). Both directed acyclic graphs have the same number of variables and the same arcs. But let us consider the dependencies of these GBNs.

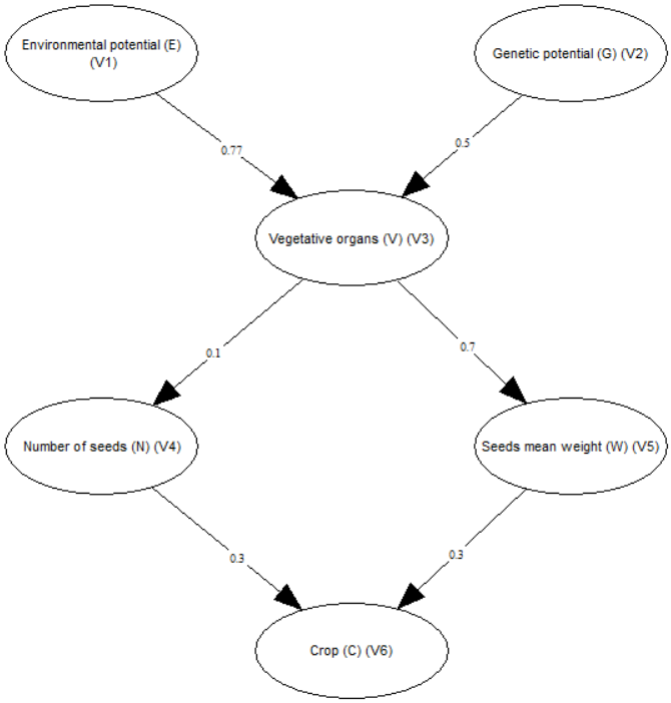


Figure 3.1: A directed acyclic graph consisting of 6 variables representing the "crop analysis".

3.2 Dependencies in GBNs

Just as we did for discrete Bayesian networks we can describe the joint distribution over all variables as multiple marginal distributions. This will give the following joint distribution for the "crop analysis".

$$\begin{aligned} P(E,G,V,N,W,C) &= \prod_{v\in\{E,G,V,N,W,C\}} P(v|E,G,V,N,W,C) \\ &= P(E)\ P(G)\ P(V|E,G)\ P(N|V)\ P(W|V)\ P(C|V,W) \end{aligned} \tag{3.2}$$

This will make the math less cumbersome. For the variables described in figure 3.1 we got the distributions in equation 3.3.

$$\begin{aligned} G &\sim \mathcal{N}(50, 10^2) \\ E &\sim \mathcal{N}(50, 10^2) \\ V|G = g, E = e &\sim \mathcal{N}(-10.35534 + 0.5g + 0.70711, 5^2) \\ N|V = v &\sim \mathcal{N}(45 + 0.1v, 9.949874^2) \\ W|V = v &\sim \mathcal{N}(15 + 0.7v, 7.141428^2) \\ C|N = n, W = w &\sim \mathcal{N}(0.3n + 0.7w, 6.25^2) \end{aligned} \tag{3.3}$$

All these variables are normally distributed and notice that for each node only their parental nodes affect the mean, just as assumed at the beginning of chapter 3. Furthermore, these dependencies only have an additive effect on the child node, which will be convenient for their mathematical properties. This may seem like a drastic assumption, but we are able to approximate small variations for any continuous function with the use of the first-order Taylor expansion, also known as linearization. Besides, relatively simple models will often perform better than more sophisticated ones, when considering less data is available (Scutari and Denis, 2014). The dependencies of GBNs are described with the use of correlation coefficients. We will give a formula for the correlation coefficients in 3.3.2.

Because of the fourth assumption that every marginal distribution of each variable can be expressed as a Gaussian linear model, we will get the following equations:

$$\begin{aligned} V &\sim \mu_V + E\beta_E + G\beta_G + e_V \\ N &\sim \mu_N + V\beta_V + e_N \\ W &\sim \mu_W + V\beta_V + e_W \\ C &\sim \mu_C + N\beta_N + W\beta_W + e_C \end{aligned} \tag{3.4}$$

With μ the intercept for a certain variable and β the correlation coefficient for a certain variable. These correlation coefficients are also seen in figure 3.1. With these equations, we are able to set up learning algorithms to learn the parameters of our GBNs.

3.3 Learning

As introduced we will first look at the parameter learning algorithms and afterwards, we will look at how the structure learning algorithms.

3.3.1 Parameter learning

We will introduce two learning algorithms.

- *Maximum likelihood*
- *Ridge regression*; this ridge regression model is generally better than the ordinary least squares model in prediction (Qshick, 2019). The least squares model is used to estimate the β in 3.4

Let us specify observations for a certain variable by $x_{i,V}$ for $i = 1, \ldots, n$ with V representing one of the variables in our network.

Maximum likelihood

Suppose we would like to learn the parameters of our variable C with the use of maximum likelihood. We know we got the following linear regression model:

$$C \sim \mu_C + N\beta_N + W\beta_W + e_V \tag{3.5}$$

As in 3.4. To make the mathematics less cumbersome we will apply the log-likelihood function, which will give us.

$$\begin{aligned}(\mu_C, \beta_N, \beta_W, \sigma_C) \mapsto & log \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{1}{2}(C_i - \mu_{i,C} - \beta_N x_{i,N} - \beta_W x_{i,W})^2/\sigma_C^2 \\ & = -\frac{1}{2} n \ log \ 2\pi - \frac{1}{2} n \ log \ \sigma_C^2 - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (C_i - \mu_{i,C} - \beta_N x_{i,N} - \beta_W x_{i,W})^2\end{aligned} \tag{3.6}$$

Because we are interested in maximum likelihood we will take the partial derivatives and set them equal to zero. We will call our obtained function f which will give us

$$\begin{aligned}
\frac{\partial f}{\partial \mu_C} &= \frac{n}{\sigma_C^2}\sum_{i=1}^{n}(C_i - \mu_{i,c} - \beta_N x_{i,N} - \beta_W x_{i,W}) \\
\frac{\partial f}{\partial \beta_N} &= \frac{n}{\sigma_C^2}\sum_{i=1}^{n}(C_i - \mu_{i,c} - \beta_N x_{i,N} - \beta_W x_{i,W})x_{i,N} \\
\frac{\partial f}{\partial \beta_W} &= \frac{n}{\sigma_C^2}\sum_{i=1}^{n}(C_i - \mu_{i,c} - \beta_N x_{i,N} - \beta_W x_{i,W})x_{i,W} \\
\frac{\partial f}{\partial \sigma_C} &= -\frac{1}{2}\sigma_C^2 - \frac{1}{2\sigma_C^4}\sum_{i=1}^{n}(\mu_{i,c} - \beta_N x_{i,N} - \beta_W x_{i,W})^2
\end{aligned} \tag{3.7}$$

Setting the partial derivatives of μ_C, β_N and β_W equal to zero will give us:

$$\begin{aligned}
\sum_{i=1}^{n}(C_i - \hat{\mu}_{i,c} - \hat{\beta}_N x_{i,N} - \hat{\beta}_W x_{i,W}) &= 0 \\
\sum_{i=1}^{n}(C_i - \hat{\mu}_{i,c} - \hat{\beta}_N x_{i,N} - \hat{\beta}_W x_{i,W})x_{i,N} &= 0 \\
\sum_{i=1}^{n}(C_i - \hat{\mu}_{i,c} - \hat{\beta}_N x_{i,N} - \hat{\beta}_W x_{i,W})x_{i,W} &= 0
\end{aligned} \tag{3.8}$$

Let us introduce two new variable $\boldsymbol{\beta} = (\beta_N, \beta_W)$ and $X = (\boldsymbol{x_N}, \boldsymbol{x_W})$. These equation will eventually give us the following maximum likelihood estimators

$$\begin{aligned}
\hat{\mu}_c &= \bar{C} - \hat{\beta_N}\bar{x_N} - \hat{\beta}_W \bar{x}_W \\
\hat{\boldsymbol{\beta}} &= (X^T X)^{-1} X^T Y
\end{aligned} \tag{3.9}$$

Suppose we would not have assumed that the conditioning effect of the parent node is given by additive linear term, then these calculations would be a lot harder. Nevertheless, the maximum likelihood estimators are still quite cumbersome to calculate for every node. Luckily, there is a function in R that can do the math for us. So let us consider the other parameter learning algorithm.

Ridge regression

Let us again consider the variable C. The estimators for β_N and β_W can also be calculated with the use of the *ridge regression* method.

$$\{\hat{\beta}_N^{RIDGE}, \hat{\beta}_W^{RIDGE}\} = \underset{\beta_N, \beta_W}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} (x_{i,c} - \mu_c - x_{i,N}\beta_N - x_{i,W}\beta_W)^2 + \lambda_2(\beta_N^2 + \beta_W^2) \right\} \tag{3.10}$$

Which is a specific form of the *elastic net*, which looks like this.

$$\{\hat{\beta}_N^{RIDGE}, \hat{\beta}_W^{RIDGE}\} = \underset{\beta_N, \beta_W}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} (x_{i,c} - \mu_c - x_{i,N}\beta_N - x_{i,W}\beta_W)^2 + \lambda_1(|\beta_N| + |\beta_W|) + \lambda_2(\beta_N^2 + \beta_W^2) \right\} \tag{3.11}$$

We see we will acquire the ridge regression if we set $\lambda_1 = 0$. λ_1 and λ_2 are used to reduce β_N and β_W. So when λ increases, β will decrease. This will prevent extreme values for β, so smoother estimators, which will lead to better predictive power (Scutari and Denis, 2014).

The reason we will be using ridge regression instead of elastic or even lasso is (*Lasso vs Ridge vs Elastic Net | ML* 2020):

- Lasso tends to make coefficients equal to absolute zero as compared to Ridge which never sets the value of coefficient to absolute zero. If the number of predictors is greater than the number of observations, Lasso will pick at most n predictors as non-zero, even if all predictors are relevant (or may be used in the test set). Of course, this will mean a lot of predictors and a few observations, which we will not consider. Nevertheless, this makes ridge regression more applicable in multiple situations.
- If there are two or more highly collinear variables then lasso regression select one of them randomly which is not good for the interpretation of data

And because elastic net makes use of ridge regression and lasso, we will make use of only ridge regression.

3.3.2 Structure learning

Just as with discrete Bayesian networks there two classes of criteria used to learn the structure; conditional independence tests and network scores. We also make the same assumptions as in 2.3.1.

Conditional independence tests

We will look at two learning algorithms with conditional independence tests criteria, just like in the discrete situation where we considered the mutual information test and log-likelihood ratio test. The most common test for GBNs is the *exact test* for *partial correlations*. The *Pearson product-moment correlation coefficient*, also known as just *Pearson correlation*, given in 3.12

$$\rho_{C,W} = COR(C,W) = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_{i,C} - \bar{x}_C)(x_{i,W} - \bar{x}_W)}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{i,C} - \bar{x}_C)^2}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{i,W} - \bar{x}_W)^2}} \tag{3.12}$$

can only express marginal linear dependencies between two variables, like in this case for the variables C and W (Scutari and Denis, 2014)(Bhatta, 2017).

However, we are interested in conditional dependencies. Consider the following null hypothesis that C is independent of W given N

$$H_0 : C \perp\!\!\!\perp_P W \mid N \tag{3.13}$$

This will be our null hypothesis of independence and our alternative hypothesis will look like

$$H_1 : C \not\perp\!\!\!\perp_P W \mid N \tag{3.14}$$

This will implicate that we would set $\beta_W = 0$ in equation 3.5. The correlation we need to test is the partial correlation between C and W, given N. So let us introduce a new correlation coefficient, $\rho_{C,W|N}$, with

$$C \perp\!\!\!\perp_P W \mid N \iff \rho_{C,W|N} \approx 0 \tag{3.15}$$

We also know that

$$\beta_W = 0 \iff \rho_{C,W|N} \approx 0 \tag{3.16}$$

This holds for all correlation coefficients for any variable. The problem is we can not express $\rho_{C,W|N}$ in a closed form, thus we need to approximate it numerically. So we need to construct a certain *correlation matrix* for C,W and N, which will look like

matrix 3.17

$$\begin{pmatrix} \rho_{C,C|N,W} & \rho_{C,W|N} & \rho_{C,N|W} \\ \rho_{W,C|N} & \rho_{W,W|C,N} & \rho_{W,N|C} \\ \rho_{N,C|W} & \rho_{N,W|C} & \rho_{N,N|C,W} \end{pmatrix} \tag{3.17}$$

Note

$$\begin{gathered} \rho_{C,C} = \rho_{W,W} = \rho_{N,N} = 1, \\ \rho_{C,W} = \rho_{W,C} \end{gathered} \tag{3.18}$$

This numerical approach can be used to test the null hypothesis. In UNINET they use a different correlation coefficient, known as *Spearman's rank correlation coefficient* or just*Spearman coefficient*. The main differences are.

- The Pearson correlation evaluates the linear relationship between two continuous variables. A relationship is linear when a change in one variable is associated with a proportional change in the other variable.
- The Spearman correlation evaluates the monotonic relationship between two continuous or ordinal variables. In a monotonic relationship, the variables tend to change together, but not necessarily at a constant rate. The Spearman correlation coefficient is based on the ranked values for each variable rather than the raw data (unknown, 2019).

For discrete BNs we have multiple possibilities to test the hypothesis, but these tests are different because of the continuous variables, so the tests will be looking at are

- *Exact-t test* for the Pearson correlation coefficient, which we just considered.
- *Fisher's Z test*, with an asymptotic normal distribution

We take n as the number of observations. The exact t-test for the Pearson correlation coefficient is defined as

$$t(X, Y|\boldsymbol{Z}) = \rho_{XY|\boldsymbol{Z}} \sqrt{\frac{n - |\boldsymbol{Z}| - 2}{1 - \rho^2_{XY|\boldsymbol{Z}}}} \tag{3.19}$$

With $n - |\boldsymbol{Z}| - 2$ degrees of freedom (Scutari and Denis, 2014).

Fisher's Z test, a transformation $\rho_{XY|\boldsymbol{Z}}$ with a asymptotic normal distribution and is defined as

$$Z(X, Y|\boldsymbol{Z}) = log\Big(\frac{1+\rho_{XY|\boldsymbol{Z}}}{1-\rho_{XY|\boldsymbol{Z}}}\Big)\frac{\sqrt{n-|\boldsymbol{Z}|-3}}{2} \tag{3.20}$$

$|\boldsymbol{Z}|$ represents the amount of variables in $\boldsymbol{Z}$.

The degrees of freedom are different compared to the discrete Bayesian networks because we do not iterate over the possible values we can obtain for the values. The distribution for the test under the null hypothesis is a student's t distribution

Network scores

These scores for GBNs are similar to discrete network scores of Bayesian networks. The BIC will be for example

$$\begin{aligned} BIC &= log\ \hat{f}(E,G,V,N,W,C) - \frac{d}{2}log\ n \\ &= \Big[log\hat{f}(E) - \frac{d_E}{2}log\ n\Big] + \Big[log\hat{f}(G) - \frac{d_G}{2}log\ n\Big] \\ &+ \Big[log\hat{f}(V|E,G) - \frac{d_V}{2}log\ n\Big] + \Big[log\hat{f}(N|V) - \frac{d_N}{2}log\ n\Big] \\ &+ \Big[log\hat{f}(W|V) - \frac{d_W}{2}log\ n\Big] + \Big[log\hat{f}(C|N,W) - \frac{d_C}{2}log\ n\Big] \end{aligned} \tag{3.21}$$

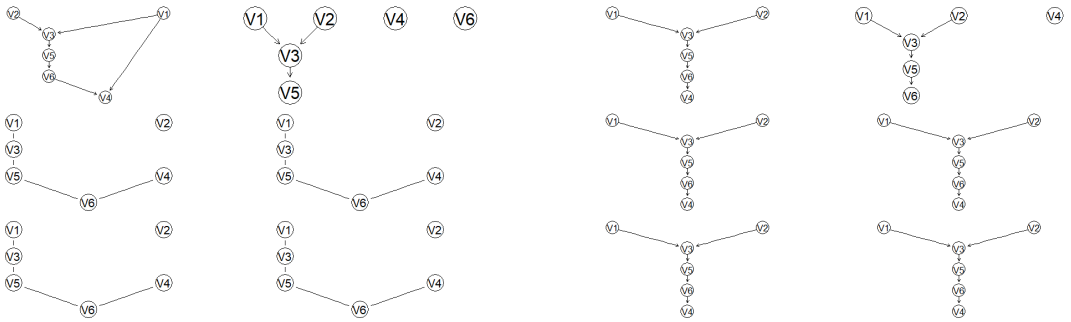
Furthermore, the structure learning algorithms (hc, gs, iamb) work exactly the same as described in 2.4.

3.4 Applying the learning algorithms

We are ready to apply the learning algorithms to our example introduced at the beginning of the chapter. Let us look at the results for multiple sizes of observations. We will first look at the results of the structure learning algorithms and after that, we will look at the parameter learning estimators.

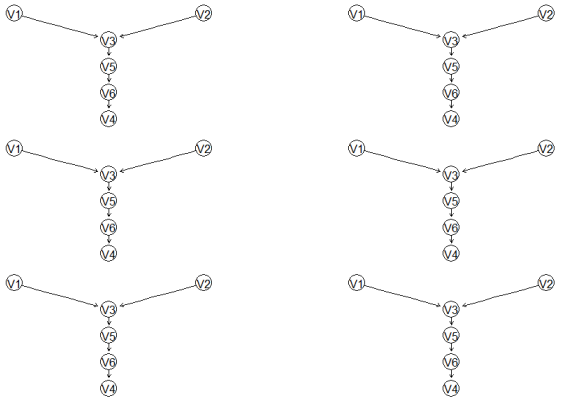
Structure learning

The structures learned by the multiple learning algorithms can be found in figure 3.2



(a) 100 observations

(b) 200 observations



(c) 400 observations

Figure 3.2: The learned structure for 100, 200 and 400 observations for discrete the crop GBN. For each figure we see from left to right on the top: hc based on BIC-g, hc based on BGe. In the middle: GS based on zf, GS based on cor. On the bottom row: iamb based on zf, iamb based on cor.

It is astonishing how much faster GBNs are learned compared to the discrete BNs, although we do see a slightly different structure, which is not changing for even more observations. We will investigate this in chapter 4. But let us consider the estimators, how well will they perform?

Parameter learning

When applying both the Ridge regression and maximum likelihood estimators to the directed acyclic graph in figure 3.1 we found the following results for 400 observations.

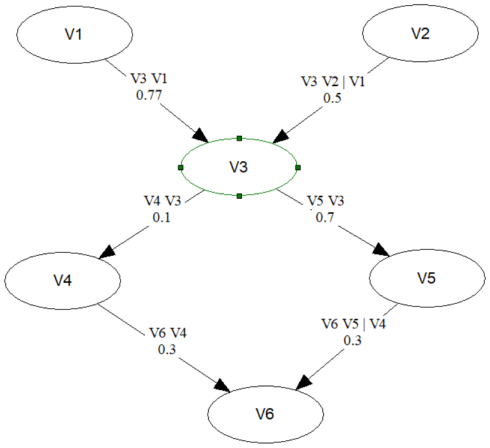


Figure 3.3: GBN of the "crop analysis" with ordering

An important note; we make use of a different correlation coefficient as discussed, so the correlation matrix is equal to

Node	**V1**	**V2**	**V3**	**V4**	**V5**	**V6**
V1	1	0	0.7711	0.0785596	0.545154	0.183357
V2	0	1	0.307083	0.0320061	0.219634	0.0746239
V3	0.7711	0.307083	1	0.1	0.7	0.233582
V4	0.0785596	0.0320061	0.1	1	0.0716577	0.3
V5	0.545154	0.219634	0.7	0.0716577	1	0.3067
V6	0.183357	0.0746239	0.233582	0.3	0.3067	1

This may seem a bit unexpected, but note that UNINET makes a distinction between which arc was added in first instance, because the conditional rank correlations are sequenced in the order in which the arcs were added. Sequencing of conditional rank correlations assures that the correlations are algebraically independent: any numbers between -1 and 1 may be assigned and will be consistent, regardless of any other correlations specified elsewhere in the BN (Dan Ababei, 2008). So with specification of the order we will get this figure 3.3. The code for parameter learning can be found in A.1.

bn.ridge represents the conditional dependency learned by ridge regression and *bn.mle* is the learned conditional dependency learned by the maximum likelihood estimator. Our implemented GBN in UNINET differs of the one we presented in this chapter, so we can not compare the parameters. In A.1.2 we can view the resulting parameter estimations.

Luckily, it is possible to redefine the correlations that such a way that it will perfectly represent the example. For this we need to define the relationship between the product moment correlation, which is used in Pearson's correlation, and rank correlation, which is used in Spearman correlation (A.M. Hanea and D.A. Ababei, 2010). The relationship between the product moment correlation (ρ) and the rank correlation (r) for joint normal, is given by *Pearson's transformation*, which we can see in 3.22

$$\rho(X, Y) = 2sin(\frac{\pi}{6}r(x, y)) \tag{3.22}$$

This can be rewritten into 3.23

$$r(x, y) = \frac{6}{\pi}sin^{-1}((\frac{1}{2}\rho)) \tag{3.23}$$

Now we know we are able to rewrite our GBN, but we will focus on the structure learning algorithm themselves.

Chapter 4

Robustness

In this chapter, we will apply all our knowledge of Bayesian networks and find the answers on how much certainty there is when learning the structures of BNs. We will first look at the discrete Bayesian networks and afterwards at the continuous Bayesian networks. The first instance will introduce the Bayesian network created with the use of Netica and Uninet. There will be an elucidation on why we choose certain Bayesian and what we hope to achieve. Then we will make our datasets by generating a sample from each of these Bayesian networks. With the datasets ready we will apply our structure learning algorithms.

In this chapter, we will make use of multiple functions and scripts in R we do need some extra clarification. The code for the results is given in A.2.

- *bnlearn::bn.cv*; performs k-fold cross-validation for a learning algorithm. We choose $k = 5$ in every situation. This means that we will divide our data into 5 random disjoints sets on which we will use 4 sets (training set) to predict the other one, the test set. We choose $runs = 20$, so we prevent extremes, which give a bad indication. We will be making box-plots of this function for every learning algorithm for a specific BN and for a specific sample size to compare the ability to predict data. We set the loss function equal to Log-Likelihood Loss.it is the negated expected log-likelihood of the test set for the Bayesian network fitted from the training set. Lower values are better (Scutari, 2020a).
- *Step functions*; we made a script that makes a step function of the learning algorithms. This script starts with one observation of the sample and learns

the structure with the use of the six possible different learning algorithms. It will add an observation and learn the structure again. It then compares the learned structures and looks if they are the same, if they are not the same the function will make a step (increase with one). This will be done for a large number of observations. So these step functions will represent the number of times the learned algorithms switch of the learned algorithm. The plot will also indicate where these switches took place.

- *bnlearn::score*; this functions calculations the score of a BN on a given dataset. It can be used for multiple scores (Scutari, 2020c) .
- *bnlearn::arc.strength*; Measure the strength of the probabilistic relationships expressed by the arcs of a Bayesian network (Scutari, 2020b).

We will also introduce another concept.

Definition 4.0.1 (Robustness) *We will consider a relative subjective interpretation of robustness of learning algorithms, where we take into account:*

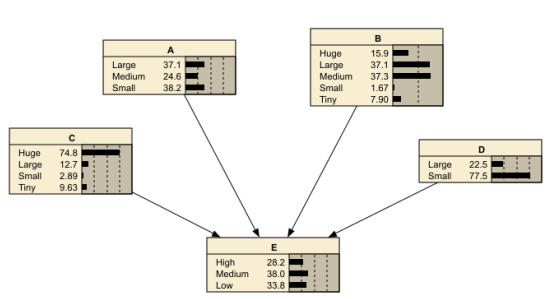
1. *The number of times a learning algorithm switches of learned structure after finding a good structure. A good learned structure has a comparable score with the true form of the structure. This is the structure on which the sample is based.*
2. *How many observations it takes for a learning algorithm to stop changing its learned algorithm. The last structure learned in an algorithm will be called final form.*
3. *the number of extreme values find when applying the cross-validation (outliers in the boxplot).*

4.1 Discrete networks

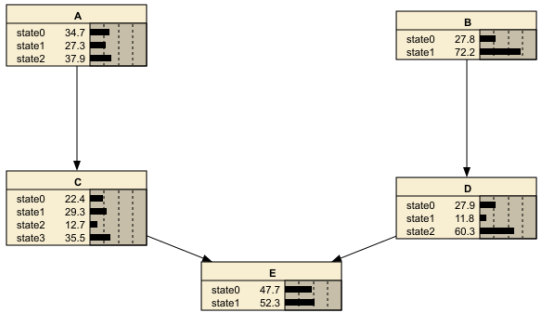
4.1.1 Randomly generated data

Our created Bayesian networks consist of approximately 5 nodes. These datasets are samples generated of a BN created with Netica. We will first introduce these BNs and specify their dependencies. Afterwards, we will apply the structure learning algorithms on the generated datasets.

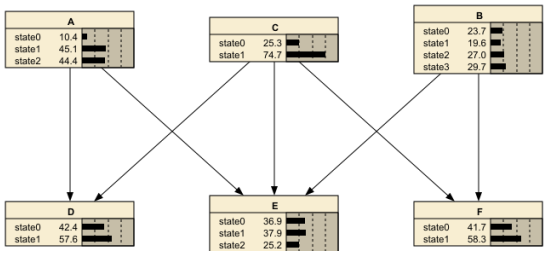
We created the following BNs with the use of Netica.



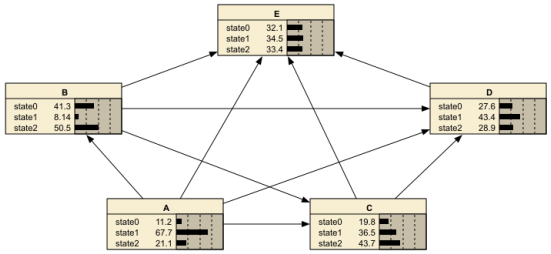
(a) Discrete BN 1



(b) Discrete BN 2



(c) Discrete BN 3



(d) Discrete BN 4

Figure 4.1: The multiple created Bayesian networks with use of Netica

We choose the BN in figure 4.1a because it is probably hard to learn. Variable E consists of an enormous probability table, which is completely random. So it is interesting to see if this makes the structure learning. figure 4.1b is a relative easy BN. There is only one v-structure and a few paths. So we expect to learn this BN rather fast. In figure 4.1c are a lot of v-structures, but only paths of 1 arc. This will probably be easy to learn. Figure 4.1d is a special BN in which all possible arcs are drawn. This will be hard to learn BN. For each of these BNs we generated a

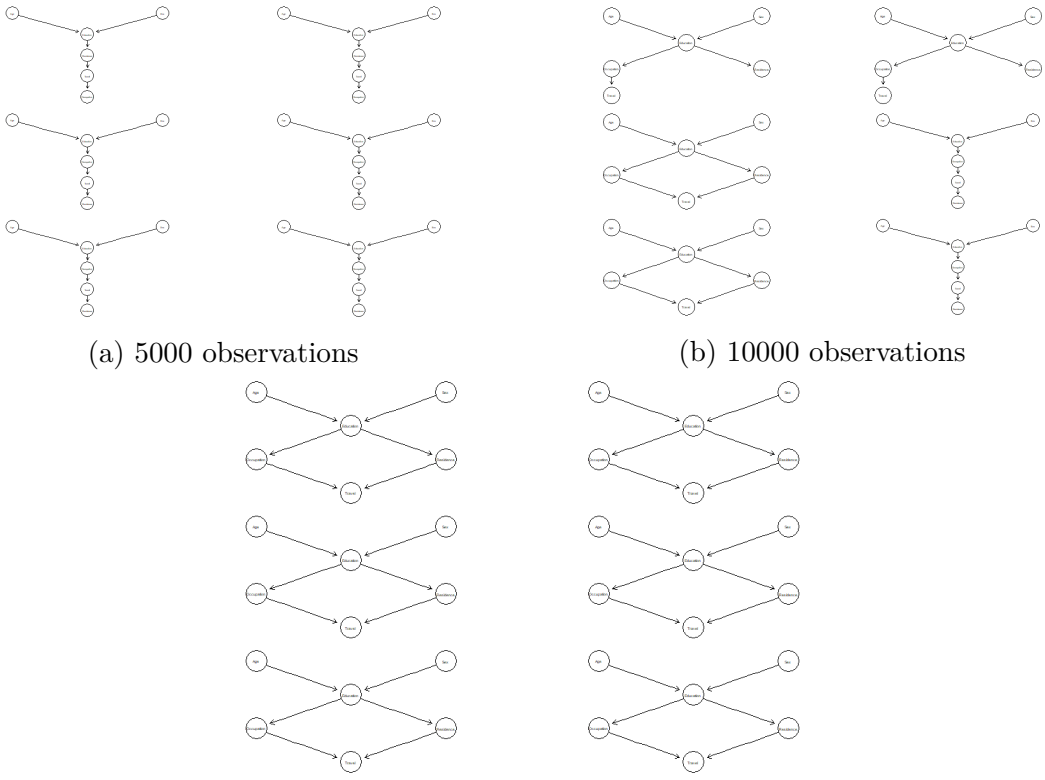
sample of a million observations. When investigating the BN we will see how many observations are needed to give a well-learned structure.

Just as discussed before, we will apply the following algorithms and their tests and scores to find the structures: *Hill climbing* based on the *BIC* score and *BDe* score, the *Grow-shrink* algorithm based on the *Pearson's* and *mutual information* test and the *Incremental association* algorithm based on the *Pearson's* and *mutual information* test.

But first we will consider the "travel use survey" example.

4.1.2 results

Travel BN



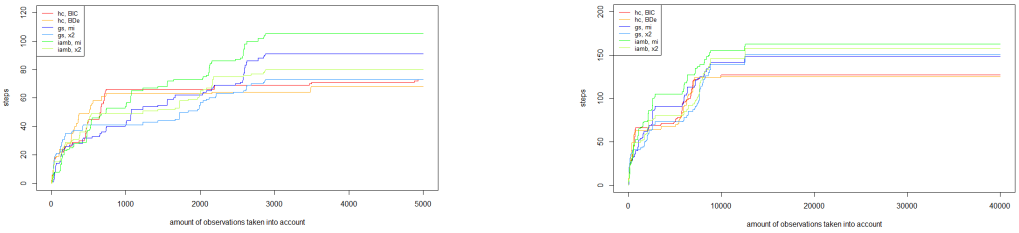
(a) 5000 observations

(b) 10000 observations

(c) 15000 observations

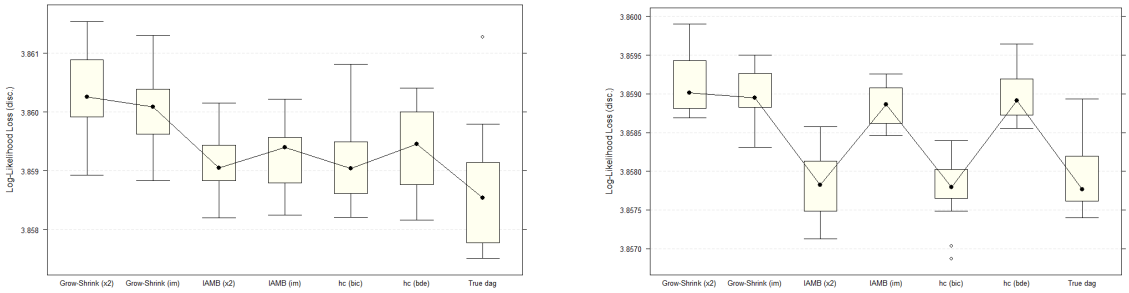
Figure 4.2: The learned structure for 5000, 10000 and 15000 observations for discrete the travel BN. For each figure we see from left to right on the top: hc based on BIC, hc based on BDe. In the middle: GS based on mi, GS based on x2. On the bottom row: iamb based on mi, iamb based on x2.

In figure 4.2, we see that after 15000 observations every learning algorithms has



(a) 5000 observations (b) 40000 observations

Figure 4.3: The step functions of the travel BN for the learned structures till 5000 and 40000 observations.



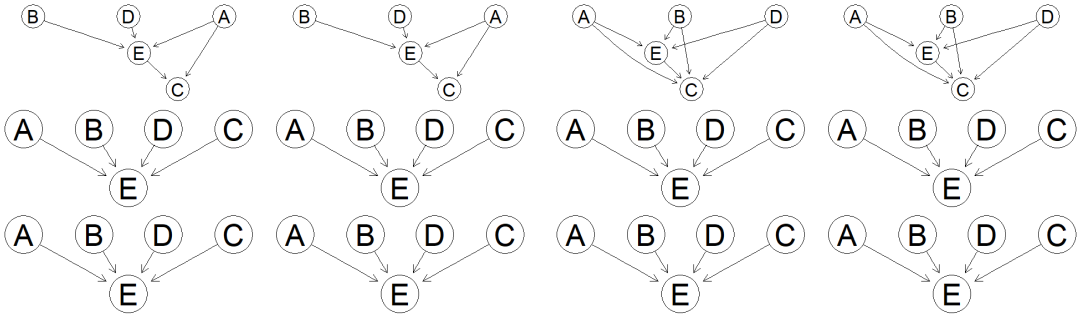
(a) 5000 observations (b) 10000 observations

Figure 4.4: The bn.cv apllied on the learned structures for 5000 and 10000 observations for discrete travel BN.

learned a structure that is equal to the one it is based on. Remarkably, every structure learning algorithm also learns the same structure after 5000 observations, but this one differs from the true form. When we look at their score we see for 5000 observations that the score of the alternative form is better, so that structure is better fitted to the data for these observations. When looking at their predictive capabilities in figure 4.4 we see that the true form performs better. When we look at the learned structures for 10000 observations we see three different structures. First, we will look closer to the ones produced by the hc algorithm. We see that there less predictive if we look at figure 4.4. When we calculate the score of the true form and the structure learned by the hc algorithm we see that the score for hc is better, so hc is better fitted to the data. When considering the structure produced by gs and iamb based on Pearson's test, we see that it is the structure as seen with 5000 observations. Only this time the score is worse than that of the true form and is less predictive, as we can see in 4.4. Looking at the step functions in figure 4.3 we

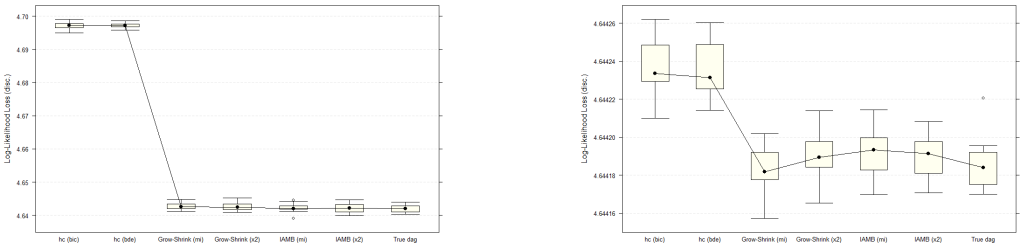
see that the hc learns the structure the fastest and is slightly more robust than the other algorithms. So overall the hc algorithm performs the best.

Discrete BN 1



(a) 10000 observations (b) 1e6 observations

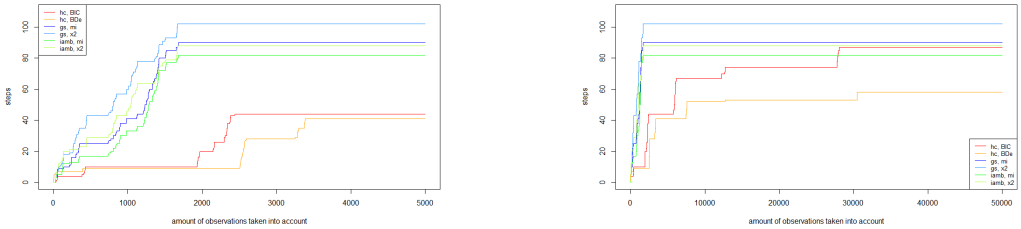
Figure 4.5: The learned structure for 1000 and 10000 observations for discrete BN 1. For both figures we see from left to right on the top: hc based on BIC, hc based on BDe. In the middle: GS based on mi, GS based on x2. On the bottom row: iamb based on mi, iamb based on x2



(a) 10000 observations (b) 1e6 observations

Figure 4.6: The bn.cv function applied on the learned structures for 10000 and 1e6 observations for discrete BN 1.

We see in 4.5 that with 10000 observations both the gs as the iamb learned a good structure, but the hc does not perform that great. We also see in the step function that iamb and gs already achieved a final form before the 2000 observations, while the hc algorithm has not achieved a final form even after 20000 observations. This said both score functions are less negative for the hill-climbing learned structure for 10000 observations. So this implicates that the hill-climbing algorithm produces structures that are better fitted to the data, but as seen by in figure 4.6 worse in predicting. When we consider a million observations we do see some interesting structures. The hill-climbing algorithm still learns a different structure, which we

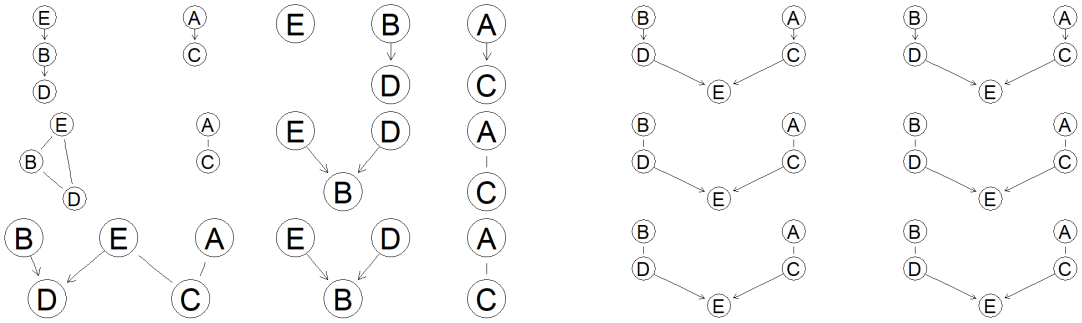


(a) 5000 observations (b) 50000 observations

Figure 4.7: Step functions for discrete BN 1

see in figure 4.6, worse in predicting. When we consider the scores of both structures we see that true form scores better. So hc produces a structure that is worse in predicting and worse fitted to the data. Gs and iamb behave similar, so we can not state a difference between the two. The hill-climbing algorithm is less robust than the other algorithms, because of the many changes. Besides we can not even tell for sure it will not change anymore for even more observations. Hill climbing does seem to behave slightly better when it is based on the BIC score.

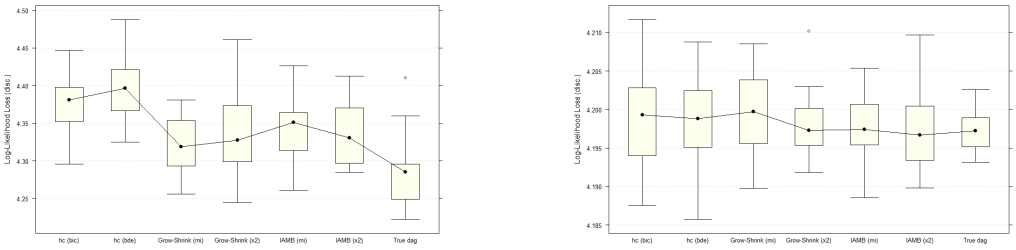
Discrete BN 2



(a) 100 observations (b) 1000 observations

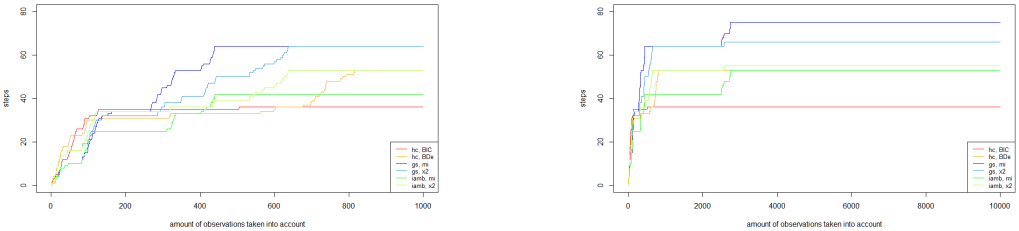
Figure 4.8: The learned structure for 100 and 1000 observations for discrete BN 2. For both figures we see from left to right on the top: hc based on BIC, hc based on BDe. In the middle: GS based on mi, GS based on x2. On the bottom row: iamb based on mi, iamb based on x2

As expected, this BN is quite easy to learn, especially for the hill-climbing algorithm. We see in figure 4.10 that for even less than 200 observations the hill-climbing does not change of learned structure. Even the learned structures after 100 observations are not that bad at all. The scores of these learned structures will not differ from each other, because they are the same, see figure 4.5. We can conclude that the hill-



(a) 100 observations (b) 1000 observations

Figure 4.9: The bn.cv function applied on the learned structures for 100 and 1000 observations for discrete BN 2.

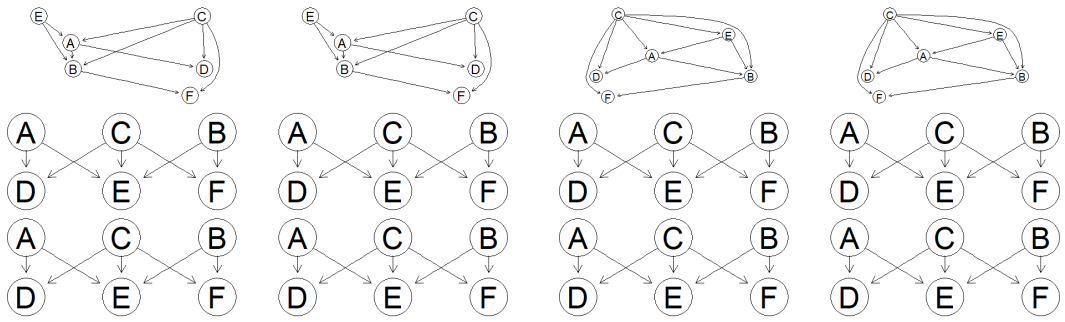


(a) 1000 observations (b) 10000 observations

Figure 4.10: The bn.cv function applied on the learned structures for 1000 and 10000 observations for discrete BN 2.

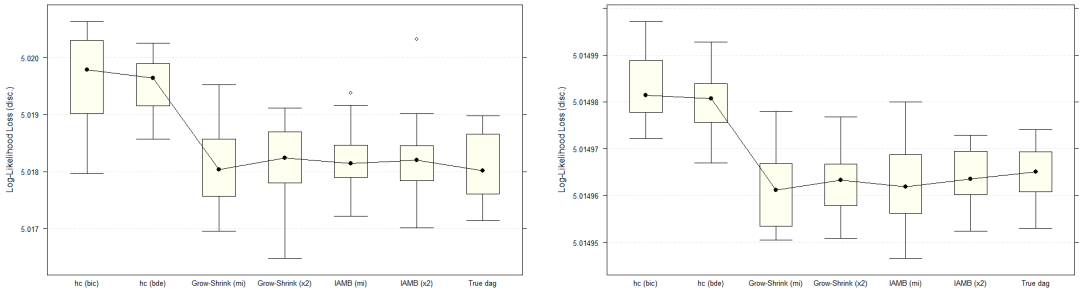
climbing algorithm is more robust and performs overall better. The hill-climbing algorithm based on the BIC score performs slightly better than based on the BDe score. Iamb is also slightly more robust than the gs algorithm because it changed less of form. We can also see both the gs and the iamb are more robust when using the mutual information test.

Discrete BN 3



(a) 10000 observations (b) 1e6 observations

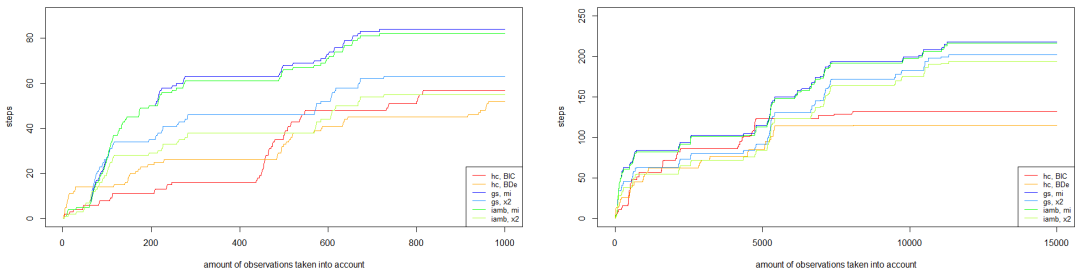
Figure 4.11: The learned structure for 10000 and 1e6 observations for discrete BN 3. For each figure we see from left to right on the top: hc based on BIC, hc based on BDe. In the middle: GS based on mi, GS based on x2. On the bottom row: iamb based on mi, iamb based on x2



(a) 10000 observations (b) 1e6 observations

Figure 4.12: The bn.cv apllied on the learned structures for 10000 and 1 million observations for discrete BN 3

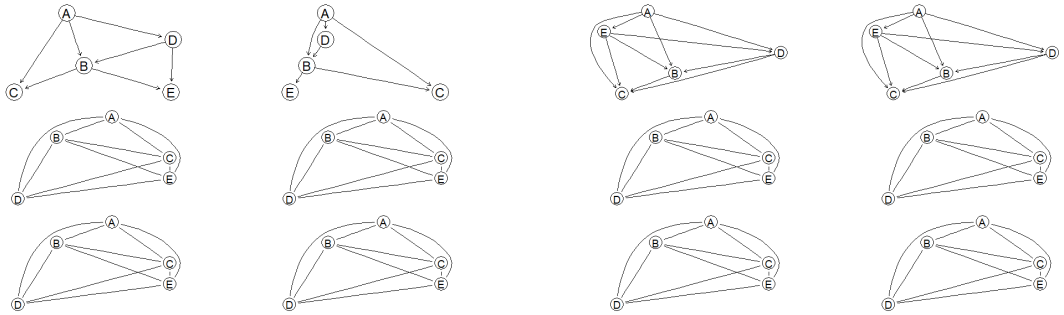
We directly notice that again the hill-climbing algorithm produces a different learned structure compared to the other algorithms and the true form. The predictive capability of hc is slightly worse for both the 10000 and 1 million observations. When we consider their scores we see that for both numbers of observations hc scores worse, so hc is less predictive and worse fitted. When looking at the step function in figure 4.13 we notice that hc does seem to be more robust. The differences between mutual information and Pearson's test are minimal, also the difference between the BIC and BDe scores is minimal. We suspected that this BN would not be too hard to learn, but it seems to be.



(a) 1000 observations (b) 15000 observations

Figure 4.13: The step functions of BN 3 for the learned structures till 1000 and 10000 observations

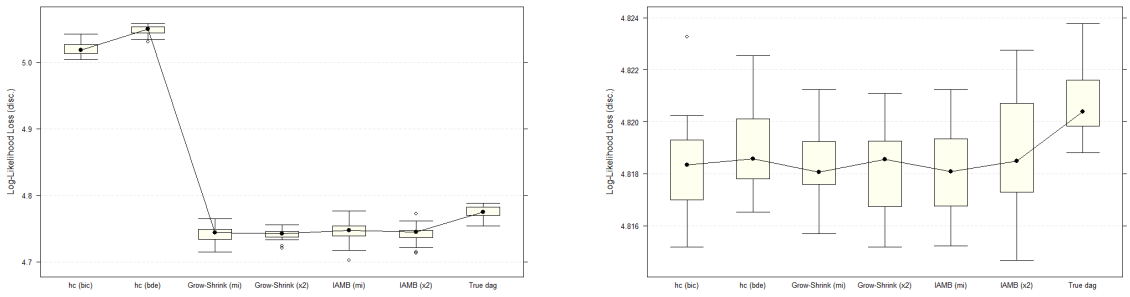
Discrete BN 4



(a) 1000 observations (b) 10000 observations

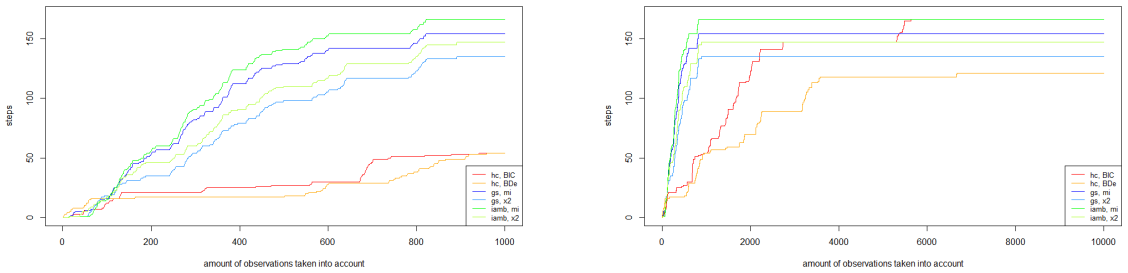
Figure 4.14: The learned structure for 1000 and 10000 observations for discrete BN 4. For each figure we see from left to right on the top: hc based on BIC, hc based on BDe. In the middle: GS based on mi, GS based on x2. On the bottom row: iamb based on mi, iamb based on x2

Beforehand we already stated that this would be a hard structure to learn, but we can see in figure 4.14 that both the gs and iamb are learning it rather fast. Again, hc takes a longer time to learn the structure and is less robust. When considering the predictive capabilities of the learned structure we notice that after 10000 observations there are actually better than the true form. This is quite surprising because this means that the true form is not the best in presenting its structure dependency. Even after 1000 observations, both gs and iamb learn better predicting structures than the true form. When we consider the score of the BNs we will see the following. For 1000 observations every structure has exactly the same score. For 10000 observations the score of hc is a lot better, than both the true form and the other algorithms. So hc learns a structure that better fits the data, but is worse in predicting data.



(a) 1000 observations (b) 10000 observations

Figure 4.15: The bn.cv apllied on the learned structures for 1000 and 10000 for discrete BN 4



(a) 1000 observations (b) 10000 observations

Figure 4.16: The step functions of BN 4 for the learned structures till 1000 and 10000 observations

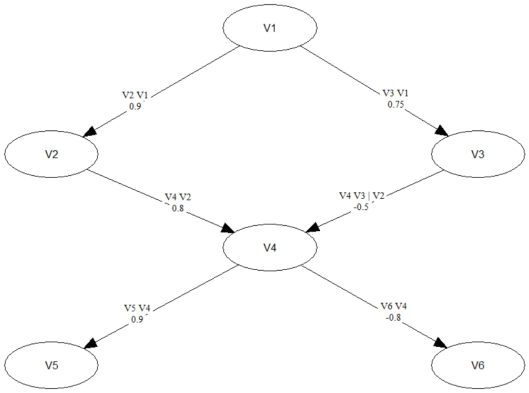
4.2 Continuous

4.2.1 Generated data

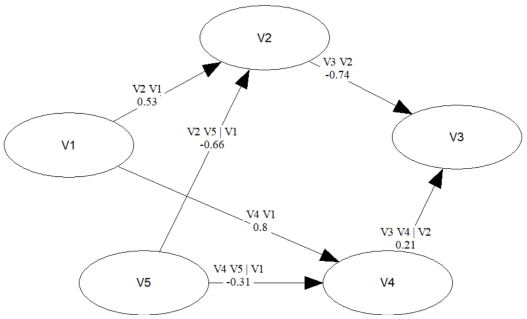
Our created Bayesian networks consist of approximately 5 nodes. These datasets are samples generated of a GBN created with UNINET. UNINET makes us of a Spearman correlation, so we will represent the correlations matrices to see what the correlations coefficients are. Afterwards, we will apply the structure learning algorithms on the generated datasets.

The chosen GBNs in figure 4.17 are quite similar to the ones we investigated in the discrete situation. This is, of course, on purpose. The main difference is that we did not randomize the probabilities. For the following reasons

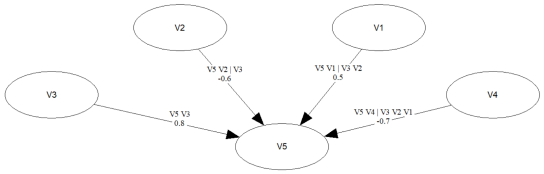
- We do not have the fill-in probability tables that could contain 360 parameters



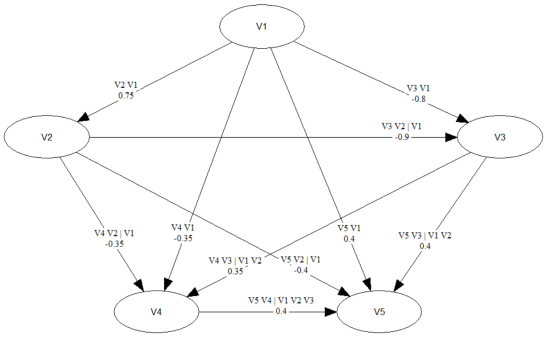
(a) GBN 1



(b) GBN 2



(c) GBN 3



(d) GBN 4

Figure 4.17: The multiple created Gaussian Bayesian networks with use of UNINET

Table 4.1: Correlation matrices

Table 4.2: Correlation matrix GBN 1

	V1	**V2**	**V3**	**V4**	**V5**	**V6**
V1	1	0.9	0.75	0.67	0.60	-0.54
V2	0.9	1	0.68	0.8	0.72	-0.64
V3	0.75	0.68	1	0.34	0.3	-0.27
V4	0.67	0.8	0.34	1	0.9	-0.8
V5	0.60	0.72	0.3	0.9	1	-0.72
V6	-0.54	-0.64	-0.27	-0.8	-0.72	1

Table 4.3: Correlation matrix GBN 2

	V1	**V2**	**V3**	**V4**	**V5**
V1	1	0.53	-0.31	0.8	0
V2	0.53	1	-0.74	0.53	-0.55
V3	-0.31	-0.74	1	-0.29	0.43
V4	0.8	0.53	-0.29	1	-0.18
V5	0	-0.55	0.43	-0.18	1

Table 4.4: Correlation matrix GBN 3

	V1	**V2**	**V3**	**V4**	**V5**
V1	1	0	0	0	0.23
V2	0	1	0	0	-0.35
V3	0	0	1	0	0.8
V4	0	0	0	1	-0.27
V5	0.23	-0.35	0.8	-0.27	1

Table 4.5: Correlation matrix GBN 4

	V1	**V2**	**V3**	**V4**	**V5**
V1	1	0.75	-0.8	-0.35	0.4
V2	0.75	1	-0.96	-0.48	0.07
V3	-0.8	-0.96	1	0.54	-0.05
V4	-0.35	-0.48	0.54	1	0.32
V5	0.4	0.07	-0.05	0.32	1

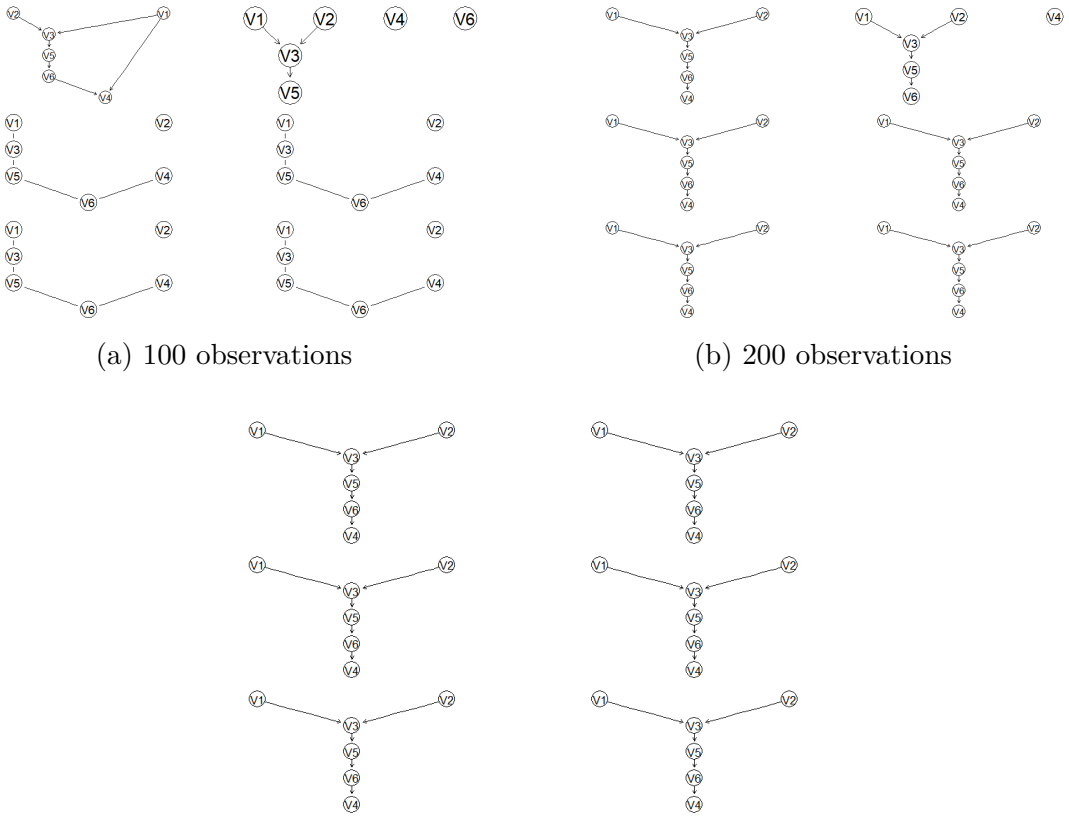
- We can make the structures more logical
- We can choose significant dependencies. In the discrete situation, it could have been possible that dependency was really weak

We choose figure 4.17a with relative strong direct dependencies, except one. It will be interesting to see if that certain arc between the nodes $V3$ and $V4$ is really harder to learn. In figure 4.17b we have chosen the variables in such a way that there are a lot of v-structures, which was hard to learn in the discrete situation. Figure 4.17c has a structure that was extremely hard to learn in the discrete situation. One dependency especially strong, to see if a high dependency is learned faster. And yet again, we have chosen a structure as in figure 4.17d to see if this structure is a bad representation of the data. We chose the dependencies in such a way that a lot of variables have a positive as negative indirect dependency on another variable. This will probably turn out to be a bad representation of the data.

Just as discussed before, we will apply the following algorithms and their tests and scores to find the structures: *Hill climbing* based on the *BIC-g* score and *BGe* score, the *Grow-shrink* algorithm based on the *linear correlation*(cor) and *Fisher's Z* (zf) test and the *Incremental association* algorithm based on the *linear correlation* and *Fisher's Z* test.

4.2.2 results

Crop analysis GBN



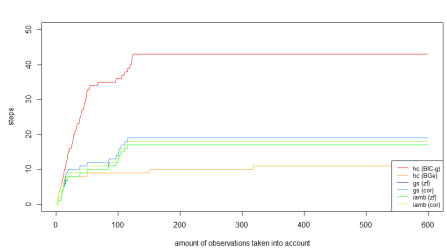
(a) 100 observations

(b) 200 observations

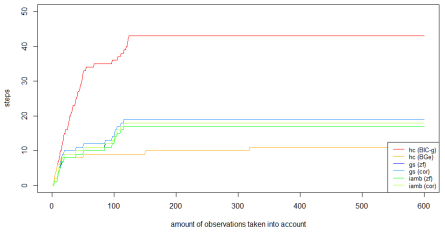
(c) 400 observations

Figure 4.18: The learned structure for 100, 200 and 400 observations for discrete the crop GBN. For each figure we see from left to right on the top: hc based on BIC-g, hc based on BGe. In the middle: GS based on zf, GS based on cor. On the bottom row: iamb based on zf, iamb based on cor.

We immediately notice in figure 4.18 that the structure is learned faster compared to the discrete BN. Almost every learning algorithm found a good structure within 200 observations. Only the hill-climbing algorithm based on the BGe score is not great. When looking at the prediction capabilities in figure 4.20 we see that the alternative structure of hill-climbing based on BGe after 200 observations is slightly worse in predicting. What is quite surprising is that the learned structure of hill-climbing after 100 observations based on BIC-g is better predicting the data. This could be due to the relative small observation size. Also, the other learning algorithms find with 200 observations another structure then the true form and their predicting

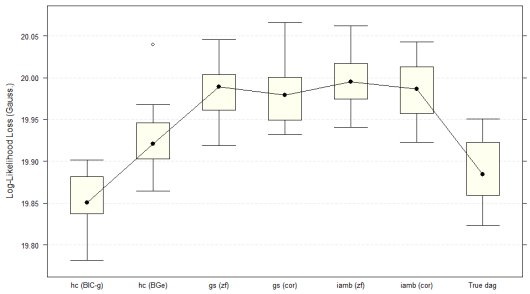


(a) 600 observations

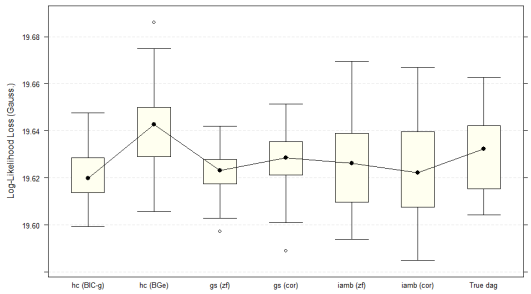


(b) 600 observations with every learned structure moralized

Figure 4.19: The step functions of BN 4 for the learned structures till 600 observations. The left without moralized learned structures. The right one with moralized learned structures



(a) 100 observations

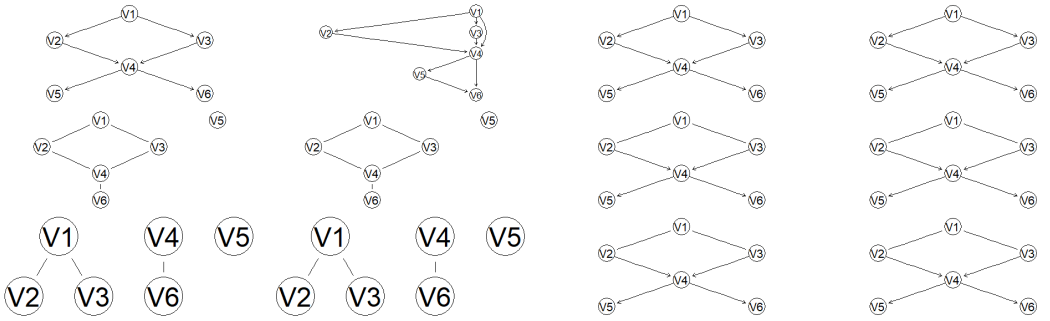


(b) 200 observations

Figure 4.20: The bn.cv apllied on the learned structures for 100 and 200 observations for discrete travel BN.

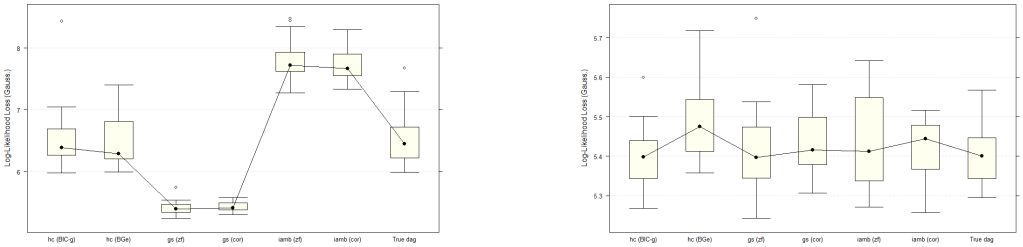
capabilities are also slightly better. When observing the scores, we found that hc algorithms are better fitted to the data for 100 observations. For 200 observations all structures that were found score better than the true form. This implies that the dependencies of $V4$ are so weak they could be neglected. This means that the true form is worse in predicting and is worse fitted to the data compared to the learned structures for 200 observations (except hc based on the BGe score, which is worse in predicting). When looking at the step functions in figure 4.19 we surprisingly enough see that hc algorithms are less robust. It may be due to the fact that the hc algorithms learn BNs and not moral graphs like the other learning algorithms, so it could be equivalent forms. We, therefore, did the step function again, but this time we compared the moral graphs of the learned structure of hc. The effect was marginal. The difference is a few steps, so it is not due to equivalent forms.

GBN 1



(a) 20 observations (b) 50 observations

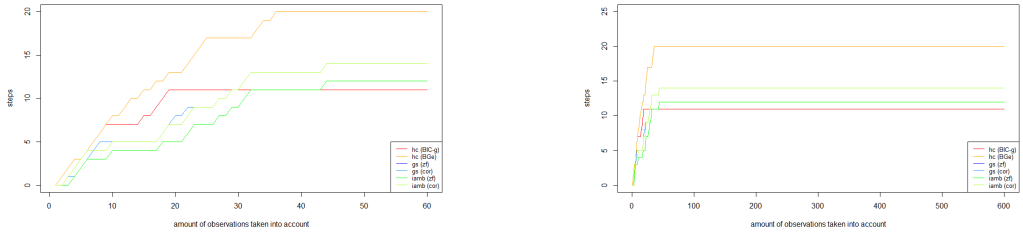
Figure 4.21: The learned structure for 20 and 50 observations for GBN 1. For each figure we see from left to right on the top: hc based on BIC-g, hc based on BGe. In the middle: GS based on zf, GS based on cor. On the bottom row: iamb based on zf, iamb based on cor



(a) 20 observations (b) 50 observations

Figure 4.22: The bn.cv function applied on the learned structures for 20 and 50 observations for GBN 1.

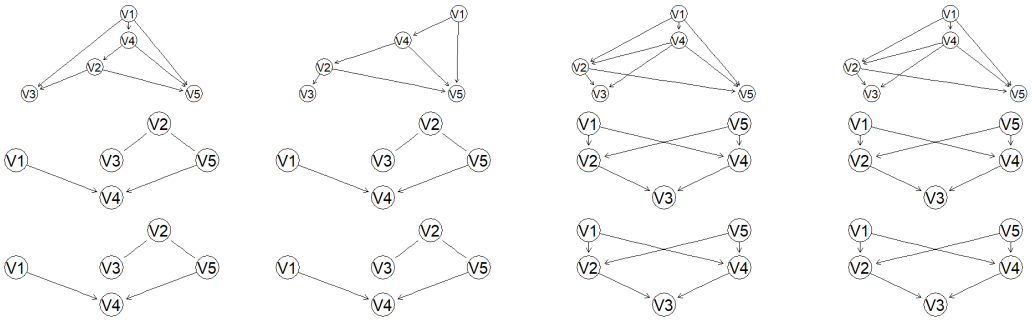
This easy to learn GBN was already learned within 50 observations for every learning algorithm. Especially the hc based on the BIC-g learned really fast. In figure 4.22 we see that gs learning algorithm is better in predicting than the true form, but this is due to the few observations we have. We see that all algorithms will eventually learn the same structure. Even with more observations, we see in figure 4.23 that the learned structure does not change after 50 observations. This also implies that all the dependencies in the structure are significant enough to be learned.



(a) 60 observations (b) 600 observations

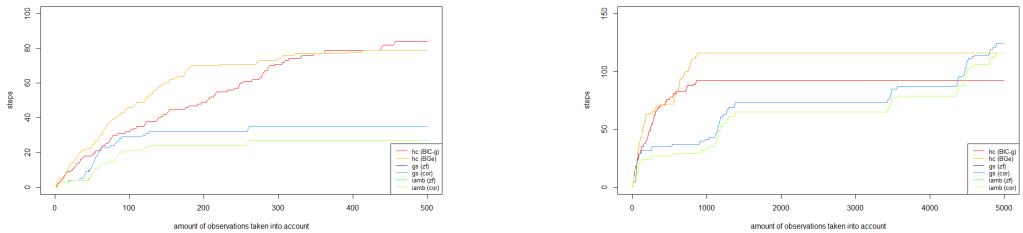
Figure 4.23: Step functions for GBN 1

GBN 2



(a) 200 observations (b) 1000 observations

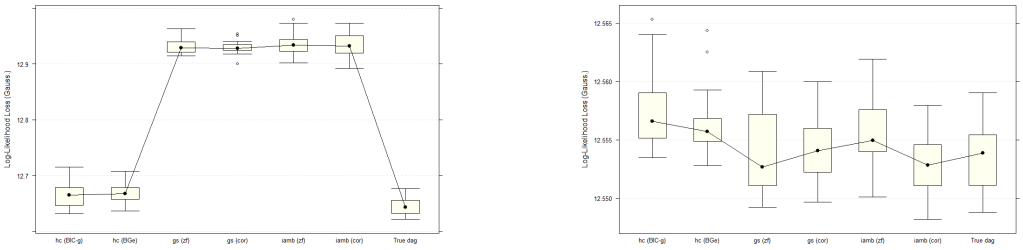
Figure 4.24: The learned structure for 200 and 1000 observations for GBN 1.



(a) 500 observations (b) 5000 observations

Figure 4.25: Step functions for GBN 2

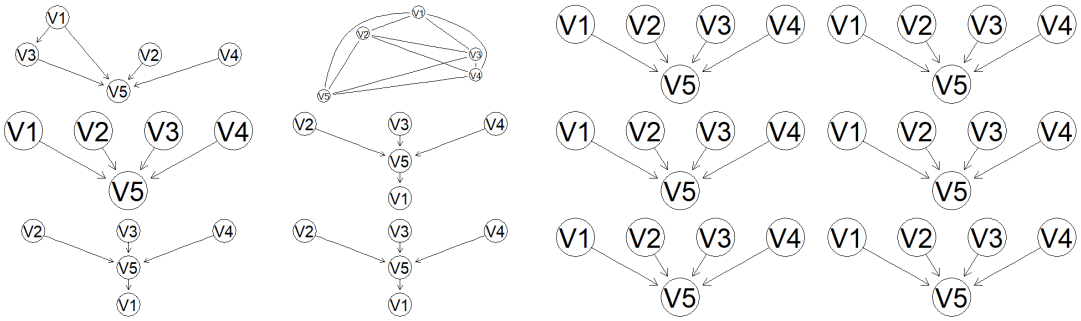
In figure 4.24 we see that this GBN is a lot harder to learn. Interesting to see is that the hc algorithm came with a different structure after 1000 observations. In figure 4.26 we see that this different structure is slightly worse in predicting the data. If we look into their scores we find that the scores of the hc algorithms are worse than that of the true form and the other algorithms. So hc is less predictive and is a worse fit to the data. This said, if we look at the step functions in figure 4.25 we see that iamb and gs are not robust at all and hc is.



(a) 200 observations (b) 1000 observations

Figure 4.26: The bn.cv function applied on the learned structures for 200 and 1000 observations for GBN 2.

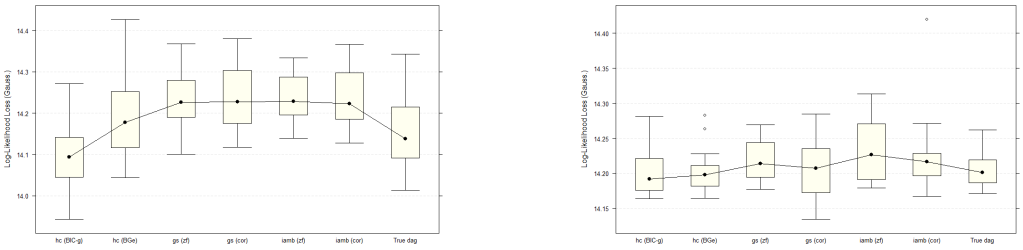
GBN 3



(a) 50 observations (b) 100 observations

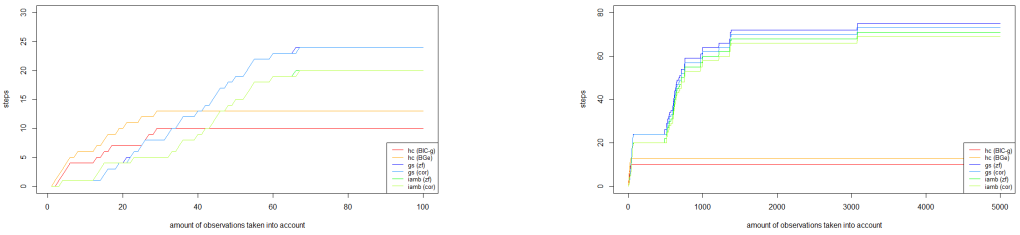
Figure 4.27: The learned structure for 50 and 100 observations for GBN 3.

In figure 4.27 we see that after 100 observations all learning algorithms find a good structure. This is quite surprising because in the discrete situation this was a really hard structure to learn. In a continuous situation, it is actually easy to learn. This may give the idea the v-structures are hard to learn in discrete BNs but are relatively easy in GBNs. Another thing that is quite noticeable, in figure 4.29 we see that the hc algorithm is more robust than the other learning algorithms. It is actually strange that even after thousands of observations the gs and iamb algorithm still change of structure. So we will take a closer look at how this alternative form looks. We see that this alternative form has an extra undirected arc between $V3$ and $V4$. It is interesting to look at the arc ($V3$ to $V4$) strength for the multiple observations it switches. It becomes clear that the threshold is 0.05 in R. So we this number is exceeded it will remove the arc. So we know this arc is circulating around this



(a) 50 observations (b) 100 observations

Figure 4.28: The bn.cv function applied on the learned structures for 200 and 1000 observations for GBN 3.

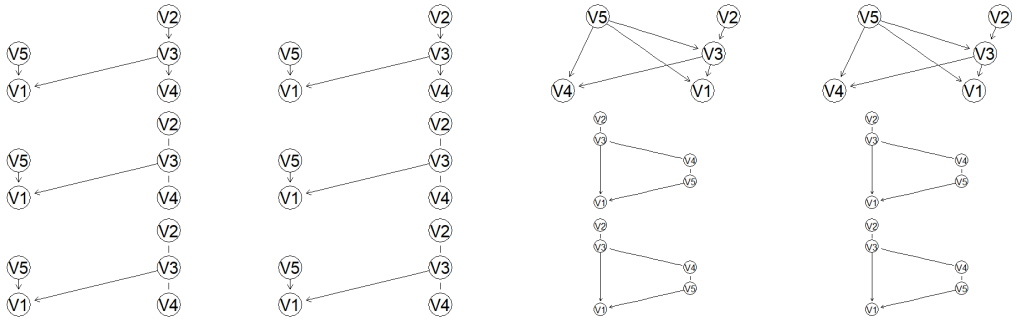


(a) 100 observations (b) 5000 observations

Figure 4.29: Step functions for GBN 3

threshold. This alternative form is worse fitted to the data because its scores are worse. Also, the predictive capability is slightly worse.

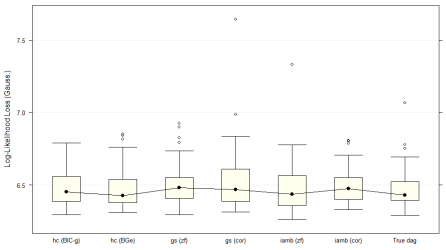
GBN 4



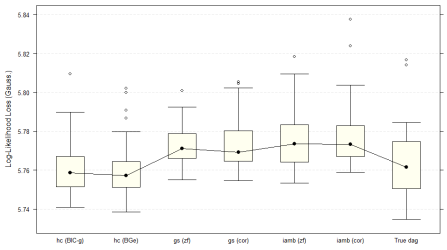
(a) 100 observations (b) 1000 observations

Figure 4.30: The learned structure for 100 and 1000 observations for GBN 4.

As expected we find different structures in figure 4.30 for the learning algorithm then the true form. When considering their predicting capabilities we notice that there quite similar, except for the structure learned for 1000 observations by the gs and iamb algorithm. When we look at their scores we found the following. The

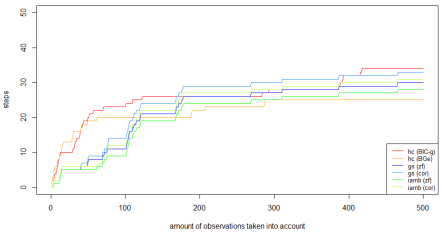


(a) 100 observations

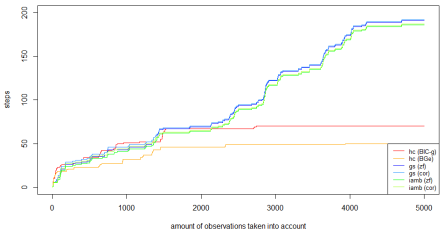


(b) 1000 observations

Figure 4.31: The bn.cv function applied on the learned structures for 100 and 1000 observations for GBN 4.



(a) 500 observations



(b) 5000 observations

Figure 4.32: Step functions for GBN 4

hc algorithm is better fitted to the data than the true form and the gs and iamb algorithm are worse fitted to the data. So hc is a better predictor and a better fit for the data. We also notice that iamb and gs are less robust than hc.

Chapter 5

Conclusions

In the literature, it is known that score-based algorithms are less sensitive to individual errors and more accurate than constraint-based algorithms, although lately there is some discussion about this statement(Marco Scutari and Gutiérrez, 2018). We will join the discussion and let our light shine on the findings we had.

We have seen a lot of different Bayesian networks and applied multiple learning algorithms on the samples of these Bayesian networks. But what did we found? Let us first discuss the discrete BN and how well our learning algorithms performed. In the most logical example, the "travel use survey", we found that the hc algorithm learned the fastest and was the most robust. However, in structures with randomized probabilities, we found something else. In most examples, the score based algorithm was the least successful in predicting new observations and seem to be rather frail for relative complex BNs. These complex BNs consist of a lot of v-structures and have a lot of direct dependencies. So it seems that for BN that are expected to have a lot of v-structures that constrained based algorithms are a better pick. For relative simple BNs, we suggest applying a score based learning algorithm.

Let us consider the GBNs. For GBNs we clearly see that the hill-climbing algorithm outperforms the other learning algorithms. The hill-climbing algorithm is more robust and learned the structures relative fast. In most examples, the constrained based algorithms are extremely frail. This is quite surprising because this is completely opposing to their performance in discrete BNs in which they showed to be more robust. Another interesting notice was that constrained based learning al-

gorithms more often learned the true form of a BN, while the score based algorithms more easily diverge from the true form.

When learning a BN it would be better to consider multiple sizes of observations, because a single observation could imply a different learned structure. Even after thousands of observations can the learned structure could well be very frail. Especially discrete BNs are harder to learn, which indicates that discretization of data will be not ideal when trying to learn a BN (John and Langley, 1995).

We also found some overall performances of these learning algorithms regarding predicting and the goodness of fit. We have seen that score based functions are usually better fitted to the data, while the constrained based algorithms are often better in predicting. This should be considered while learning because a well fitted BN does not imply a good predictor. Which makes us wonder what we want from a BN. The BN from which we take a sample is not necessarily the best BN for the representation of the dependencies captured in the sampled dataset. We already found in the results that some BNs from which we took samples were not good representations of the dependencies that it tried to show. We found that our learning algorithms were able to find structures that were a better fit for the data and could predict new observations better.

Chapter 6

Discussion

We have done a lot of research and found some remarkable aspects of BNs. Nevertheless, our goal was to find out how robust BNs structures are. We asked ourselves how certain the learned structures are and how they would react to new observations. We found that for multiple examples the learning algorithms could switch between structures even after tens of thousands of observations. But along the way, we made assumptions and simplified situations.

We tested the null hypothesis of independence only with the asymptotic chi-squared distribution, which requires an adequate sample size to be accurate. Especially for the GBNs we based a lot of our conclusions of the results on small samples. This could be a reason for the drastically worse performance of constrained based algorithms. Two common other possibilities could be considered. We could have applied the Monte Carlo permutations(Edwards, 2000), which can be applied to smaller sample sizes. Besides we could also have chosen for the semi-parametric chi-squared distribution, which will perform better on smaller sample sizes (Tsamardinos and Borboudakis, 2010).

Of course, there are also different structure learning algorithms in the literature we could have applied. When looking back, I think it would have been better to also consider another score based function. It was interesting to see the differences between both constrained based algorithms. It would have been especially interesting to investigate hybrid learning algorithms, which make use of both score based and constrained based algorithms. These algorithms seem to be the most accurate in the literature (Marco Scutari and Gutiérrez, 2018).

We did not consider the computational time of these learning algorithms, what did become a problem. For some examples, I would like to see the step functions for more observations. In some of the examples, it still was not clear if the learning algorithms would change of learned algorithm, but it already took almost half an hour to produce these step functions up to 50000 observations.

During the process, I got the feeling that the observations themselves was already too unpredictable to use to test the performance of the learning algorithms. For the GBNs we already learned good structures within 20 observations. It would be interesting to see it for multiple examples or the view the mean observations to learn an network. Some of the statements made during chapter 4 could be due to that specific observation.

Looking back at the thesis, I believe there is a possibility to make a program that can analyse the BN to see which learning algorithm would be optimal. A sort of prediction. This would mean that the BN is already known, what in practice would be unrealistic, but it could be used as a reflection to observe that the learned structure, is a structure that is harder for the current learning algorithm used, so it would be logical that a different one would learn better. Because we already saw that for some examples some learning algorithms do operate better.

During the thesis, I was trying to implement a more extensive step function that also showed the scores of the BN and their predictive capabilities. This would be extremely interesting when observing a certain number observations for which it would switch of structure (the number for which it makes a step in the step function) to see the direct effect of it on the score and the ability to predict. The problem was that the score function is constantly decreasing so certain jumps were very hard to notice. I tried to make a regression line of the score function and subtract that of the scores found, but I did not have the time to further analyse it and make it well working. For the predictive capabilities, it was even hard, because we can only estimate that ability, so we would have needed to do multiple runs for each observation. That would take a lot of computation time. For the predictive

capabilities, we only made use of cross-validation. It would be interesting to see different approaches and their outcomes.

Appendices

Appendix A

R code

A.1 Parameter estimators

```
library(bnlearn)
library(penalized)
##########parameter estimators

######travel use survey
dag = model2network("[Age][Sex][Education|Age:Sex]
    [Occupation|Education][Residence|Education]
    [Travel|Occupation:Residence]")


sample_1e6<- read.delim("~/BEP/Netica/Travel BN, discrete/
                        Travelsample_1e6.cas")[-1]
sample_15000 <- sample_1e6[0:15000,0:6]

bn.mle <- bn.fit(dag, data = sample_15000, method = "mle")

bn.bayes <- bn.fit(dag, data = sample_15000, method = "bayes",
                   iss=10)

bn.mle$Travel
```

```
bn.bayes$Travel

######crop analysis
dag <- model2network("[V1][V2][V3|V1:V2][V4|V3]
                     [V5|V3][V6|V4:V5]")

sample_1e6 <-    read.csv("~/BEP/Uninet/Crop sample/BN crop sample;
                          1e5 observations.sae")
sample_400 <- sample_1e6[0:400,0:6]

bn.mle <- bn.fit(dag, data = sample_1e6)
bn.ridge <- bn.fit(dag, data = sample_1e6)

bn.ridge$V3 <- penalized(V3 ~ V1 + V2, lambda1 = 0,
                         lambda2 = 1, data = sample_1e6)

bn.ridge$V3
bn.mle$V3
```

A.1.1 Results discrete parameter estimators

Maximum likelihood estimator

```
> bn.mle$Travel

  Parameters of node Travel (multinomial distribution)

Conditional probability table:

, , Residence = Big

        Occupation
```

```
Travel     Employee       Self
  Car    0.55971316 0.69581749
  Other  0.07869817 0.07414449
  Train  0.36158867 0.23003802

, , Residence = Small

          Occupation
Travel     Employee       Self
  Car    0.49033392 0.61748634
  Other  0.09578207 0.16939891
  Train  0.41388401 0.21311475
```

Bayes estimator

```
> bn.bayes$Travel

  Parameters of node Travel (multinomial distribution)

Conditional probability table:

, , Residence = Big

          Occupation
Travel     Employee       Self
  Car    0.55966114 0.69410281
  Other  0.07875668 0.07537055
  Train  0.36158218 0.23052665

, , Residence = Small
```

```
        Occupation
Travel     Employee       Self
  Car    0.49021904 0.61365678
  Other  0.09595590 0.17160827
  Train  0.41382506 0.21473495
```

A.1.2 Results continuous parameter estimations

```
> bn.ridge$V6

  Parameters of node V6 (Gaussian distribution)

Conditional density: V6 | V4 + V5
Coefficients:
(Intercept)            V4            V5
-12.9525012    0.1984032    0.2555957
Standard deviation of the residuals: 5.64952
> bn.mle$V6

  Parameters of node V6 (Gaussian distribution)

Conditional density: V6 | V4 + V5
Coefficients:
(Intercept)            V4            V5
-12.9529467    0.1984091    0.2556079
Standard deviation of the residuals: 5.64952
```

A.2 Learning algorithms

```
library(bnlearn)
library(Rgraphviz)
```

```
library(writexl)

sample_1e6<- read.delim("~/BEP/Netica/
                        Travel BN, discrete/Travelsample_1e6.cas")[-1]

true_dag = model2network("[Age][Sex][Education|Age:Sex]
                         [Occupation|Education][Residence|Education]
                         [Travel|Occupation:Residence]")

sample_100 <- sample_1e6[0:100,0:6]
sample_1000 <- sample_1e6[0:1000,0:6]
sample_5000 <- sample_1e6[0:5000,0:6]
sample_10000 <- sample_1e6[0:10000,0:6]
sample_15000 <- sample_1e6[0:15000,0:6]


###learning the structures for every algorithm
learn100_1 = hc(sample_100, score = "bic")
learn100_2 = hc(sample_100, score = "bde")
learn100_3 = gs(sample_100, test = "mi")
learn100_4 = gs(sample_100, test = "x2")
learn100_5 = iamb(sample_100, test = "mi")
learn100_6 = iamb(sample_100, test = "x2")

learn1000_1 = hc(sample_1000, score = "bic")
learn1000_2 = hc(sample_1000, score = "bde")
learn1000_3 = gs(sample_1000, test = "mi")
learn1000_4 = gs(sample_1000, test = "x2")
learn1000_5 = iamb(sample_1000, test = "mi")
learn1000_6 = iamb(sample_1000, test = "x2")
```

```
learn5000_1 = hc(sample_5000, score = "bic")
learn5000_2 = hc(sample_5000, score = "bde")
learn5000_3 = gs(sample_5000, test = "mi")
learn5000_4 = gs(sample_5000, test = "x2")
learn5000_5 = iamb(sample_5000, test = "mi")
learn5000_6 = iamb(sample_5000, test = "x2")

learn10000_1 = hc(sample_10000, score = "bic")
learn10000_2 = hc(sample_10000, score = "bde")
learn10000_3 = gs(sample_10000, test = "mi")
learn10000_4 = gs(sample_10000, test = "x2")
learn10000_5 = iamb(sample_10000, test = "mi")
learn10000_6 = iamb(sample_10000, test = "x2")

learn15000_1 = hc(sample_15000, score = "bic")
learn15000_2 = hc(sample_15000, score = "bde")
learn15000_3 = gs(sample_15000, test = "mi")
learn15000_4 = gs(sample_15000, test = "x2")
learn15000_5 = iamb(sample_15000, test = "mi")
learn15000_6 = iamb(sample_15000, test = "x2")

learn1e6_1 = hc(sample_1e6, score = "bic")
learn1e6_2 = hc(sample_1e6, score = "bde")
learn1e6_3 = gs(sample_1e6, test = "mi")
learn1e6_4 = gs(sample_1e6, test = "x2")
learn1e6_5 = iamb(sample_1e6, test = "mi")
learn1e6_6 = iamb(sample_1e6, test = "x2")

par(mfrow =c(3,2), mar = c(1,1,1,1))

####use cextend to get a dag from partially directed graph
```

```
graphviz.plot(learn100_1)
graphviz.plot(learn100_2)
graphviz.plot(learn100_3)
graphviz.plot(learn100_4)
graphviz.plot(learn100_5)
graphviz.plot(learn100_6)

graphviz.plot(learn1000_1)
graphviz.plot(learn1000_2)
graphviz.plot(learn1000_3)
graphviz.plot(learn1000_4)
graphviz.plot(learn1000_5)
graphviz.plot(learn1000_6)

graphviz.plot(learn5000_1)
graphviz.plot(learn5000_2)
graphviz.plot(learn5000_3)
graphviz.plot(learn5000_4)
graphviz.plot(learn5000_5)
graphviz.plot(learn5000_6)

graphviz.plot(learn10000_1)
graphviz.plot(learn10000_2)
graphviz.plot(learn10000_3)
graphviz.plot(learn10000_4)
graphviz.plot(learn10000_5)
graphviz.plot(learn10000_6)

graphviz.plot(learn15000_1)
graphviz.plot(learn15000_2)
graphviz.plot(learn15000_3)
```

```
graphviz.plot(learn15000_4)
graphviz.plot(learn15000_5)
graphviz.plot(learn15000_6)


graphviz.plot(learn1e6_1)
graphviz.plot(learn1e6_2)
graphviz.plot(learn1e6_3)
graphviz.plot(learn1e6_4)
graphviz.plot(learn1e6_5)
graphviz.plot(learn1e6_6)
graphviz.plot(true_dag)

test1 = bn.cv(sample_100, learn100_1, method = "k-fold",
              k =5, runs = 20)
test2 = bn.cv(sample_100, learn100_2, method = "k-fold",
              k =5, runs = 20)
test3 = bn.cv(sample_100, learn100_3, method = "k-fold",
              k =5, runs = 20)
test4 = bn.cv(sample_100, learn100_4, method = "k-fold",
              k =5, runs = 20)
test5 = bn.cv(sample_100, learn100_5, method = "k-fold",
              k =5, runs = 20)
test6 = bn.cv(sample_100, learn100_6, method = "k-fold",
              k =5, runs = 20)
test7 = bn.cv(sample_100, true_dag, method = "k-fold",
              k =5, runs = 20)

plot(test1 ,test2 ,test3 ,test4 ,test5 ,test6 ,test7 , connect = TRUE,
     xlab = c("Grow-Shrink (x2)", "Grow-Shrink (im)", "IAMB (x2)",
              "IAMB (im)", "hc (bic)", "hc (bde)", "True dag"))
```

```
test1 = bn.cv(sample_1000, learn1000_1, method = "k-fold",
              k =5, runs = 20)
test2 = bn.cv(sample_1000, learn1000_2, method = "k-fold",
              k =5, runs = 20)
test3 = bn.cv(sample_1000, learn1000_3, method = "k-fold",
              k =5, runs = 20)
test4 = bn.cv(sample_1000, learn1000_4, method = "k-fold",
              k =5, runs = 20)
test5 = bn.cv(sample_1000, learn1000_5, method = "k-fold",
              k =5, runs = 20)
test6 = bn.cv(sample_1000, learn1000_6, method = "k-fold",
              k =5, runs = 20)
test7 = bn.cv(sample_1000, true_dag, method = "k-fold",
              k =5, runs = 20)

plot(test1,test2,test3,test4,test5,test6,test7, connect = TRUE,
     xlab = c("Grow-Shrink (x2)", "Grow-Shrink (im)", "IAMB (x2)",
              "IAMB (im)", "hc (bic)", "hc (bde)","True dag"))

test1 = bn.cv(sample_5000, learn5000_1, method = "k-fold",
              k =5, runs = 20)
test2 = bn.cv(sample_5000, learn5000_2, method = "k-fold",
              k =5, runs = 20)
test3 = bn.cv(sample_5000, learn5000_3, method = "k-fold",
              k =5, runs = 20)
test4 = bn.cv(sample_5000, learn5000_4, method = "k-fold",
              k =5, runs = 20)
test5 = bn.cv(sample_5000, learn5000_5, method = "k-fold",
              k =5, runs = 20)
test6 = bn.cv(sample_5000, learn5000_6, method = "k-fold",
```

```
                k =5, runs = 20)
test7 = bn.cv(sample_5000, true_dag, method = "k-fold",
                k =5, runs = 20)

plot(test1 ,test2 ,test3 ,test4 ,test5 ,test6 ,test7 , connect = TRUE,
     xlab = c("Grow-Shrink (x2)", "Grow-Shrink (im)", "IAMB (x2)",
                "IAMB (im)", "hc (bic)", "hc (bde)","True dag"))


test1 = bn.cv(sample_10000, learn10000_1, method = "k-fold",
                k =5, runs = 20)
test2 = bn.cv(sample_10000, learn10000_2, method = "k-fold",
                k =5, runs = 20)
test3 = bn.cv(sample_10000, learn10000_3, method = "k-fold",
                k =5, runs = 20)
test4 = bn.cv(sample_10000, learn10000_4, method = "k-fold",
                k =5, runs = 20)
test5 = bn.cv(sample_10000, learn10000_5, method = "k-fold",
                k =5, runs = 20)
test6 = bn.cv(sample_10000, learn10000_6, method = "k-fold",
                k =5, runs = 20)
test7 = bn.cv(sample_10000, true_dag, method = "k-fold",
                k =5, runs = 20)

plot(test1 ,test2 ,test3 ,test4 ,test5 ,test6 ,test7 , connect = TRUE,
     xlab = c("Grow-Shrink (x2)", "Grow-Shrink (im)", "IAMB (x2)",
                "IAMB (im)", "hc (bic)", "hc (bde)", "True dag"))

test1 = bn.cv(sample_15000, learn15000_1, method = "k-fold",
                k =5, runs = 20)
test2 = bn.cv(sample_15000, learn15000_2, method = "k-fold",
```

```
                k =5, runs = 20)
test3 = bn.cv(sample_15000, learn15000_3, method = "k-fold",
                k =5, runs = 20)
test4 = bn.cv(sample_15000, learn15000_4, method = "k-fold",
                k =5, runs = 20)
test5 = bn.cv(sample_15000, learn15000_5, method = "k-fold",
                k =5, runs = 20)
test6 = bn.cv(sample_15000, learn15000_6, method = "k-fold",
                k =5, runs = 20)
test7 = bn.cv(sample_15000, true_dag, method = "k-fold",
                k =5, runs = 20)

plot(test1,test2,test3,test4,test5,test6,test7, connect = TRUE,
     xlab = c("Grow-Shrink (x2)", "Grow-Shrink (im)", "IAMB (x2)",
                "IAMB (im)", "hc (bic)", "hc (bde)", "True dag"))



test1 = bn.cv(sample_1e6, learn1e6_1, method = "k-fold",
                k =5, runs = 5)
test2 = bn.cv(sample_1e6, learn1e6_2, method = "k-fold",
                k =5, runs = 5)
test3 = bn.cv(sample_1e6, learn1e6_3, method = "k-fold",
                k =5, runs = 5)
test4 = bn.cv(sample_1e6, learn1e6_4, method = "k-fold",
                k =5, runs = 5)
test5 = bn.cv(sample_1e6, learn1e6_5, method = "k-fold",
                k =5, runs = 5)
test6 = bn.cv(sample_1e6, learn1e6_6, method = "k-fold",
                k =5, runs = 5)
test7 = bn.cv(sample_1e6, true_dag, method = "k-fold",
```

```
                k =5, runs = 5)


plot(test1,test2,test3,test4,test5,test6,test7, connect = TRUE,
     xlab = c("Grow-Shrink (x2)", "Grow-Shrink (im)", "IAMB (x2)",
              "IAMB (im)", "hc (bic)", "hc (bde)","True dag"))


arc.strength(learn10000_1, sample_10000)
arc.strength(true_dag, sample_10000)

bnlearn::score(learn10000_1, sample_10000, type = "bic")
bnlearn::score(learn10000_4, sample_10000, type = "bic")
bnlearn::score(true_dag,sample_10000, type = "bic")

bnlearn::score(learn10000_1, sample_10000, type = "bde")
bnlearn::score(learn10000_4, sample_10000, type = "bde")
bnlearn::score(true_dag,sample_10000, type = "bde")

bnlearn::score(learn10000_1, sample_5000, type = "bde")
bnlearn::score(learn10000_4, sample_5000, type = "bde")
bnlearn::score(true_dag,sample_5000, type = "bde")



bnlearn::score(learn10000_2, sample_10000, type = "bde")
bnlearn::score(learn10000_1, sample_10000, type = "bde")
bnlearn::score(true_dag, sample_10000, type = "bde")

###see the changes per new observation
n = 50000
```

```
changeBN_hc_BIC = c()
changeBN_hc_BDe = c()
changeBN_gs_mi = c()
changeBN_gs_x2 = c()
changeBN_iamb_mi = c()
changeBN_iamb_x2 = c()
sample = sample_1e6[0:1,0:5]
learn1_oud = hc(sample, score = "bic")
learn2_oud = hc(sample, score = "bde")
learn3_oud = gs(sample, test = "mi")
learn4_oud = gs(sample, test = "x2")
learn5_oud = iamb(sample, test = "mi")
learn6_oud = iamb(sample, test = "x2")
for(i in 1:n){
  sample = sample_1e6[0:i,0:5]
  learn1 = hc(sample, score = "bic")
  learn2 = hc(sample, score = "bde")
  learn3 = gs(sample, test = "mi")
  learn4 = gs(sample, test = "x2")
  learn5 = iamb(sample, test = "mi")
  learn6 = iamb(sample, test = "x2")
  if(!isTRUE(all.equal(learn1,learn1_oud))){
    changeBN_hc_BIC = c(changeBN_hc_BIC,i)
    learn1_oud = learn1
  }
  if(!isTRUE(all.equal(learn2,learn2_oud))){
    changeBN_hc_BDe = c(changeBN_hc_BDe,i)
    learn2_oud = learn2
  }
  if(!isTRUE(all.equal(learn3,learn3_oud))){
    changeBN_gs_mi = c(changeBN_gs_mi,i)
```

```
    learn3_oud = learn3
  }
  if(!isTRUE(all.equal(learn4,learn4_oud))){
    changeBN_gs_x2 = c(changeBN_gs_x2,i)
    learn4_oud = learn4
  }
  if(!isTRUE(all.equal(learn5,learn5_oud))){
    changeBN_iamb_mi = c(changeBN_iamb_mi,i)
    learn5_oud = learn5
  }
  if(!isTRUE(all.equal(learn6,learn6_oud))){
    changeBN_iamb_x2 = c(changeBN_iamb_x2,i)
    learn6_oud = learn6
  }
}

##step plot
a_1 = c()
b_1 = 0
a_2 = c()
b_2 = 0
a_3 = c()
b_3 = 0
a_4 = c()
b_4 = 0
a_5 = c()
b_5 = 0
a_6 = c()
b_6 = 0
for(i in 1:n){
  if(i %in% changeBN_hc_BIC){
```

```
    b_1 = b_1 + 1
  }
  if(i %in% changeBN_hc_BDe){
    b_2 = b_2 + 1
  }
  if(i %in% changeBN_gs_mi){
    b_3 = b_3 + 1
  }
  if(i %in% changeBN_gs_x2){
    b_4 = b_4 + 1
  }
  if(i %in% changeBN_iamb_mi){
    b_5 = b_5 + 1
  }
  if(i %in% changeBN_iamb_x2){
    b_6 = b_6 + 1
  }
  a_1 = c(a_1,b_1)
  a_2 = c(a_2,b_2)
  a_3 = c(a_3,b_3)
  a_4 = c(a_4,b_4)
  a_5 = c(a_5,b_5)
  a_6 = c(a_6,b_6)
}
n = 40000
graphics.off()
plot(seq(1,n), a_1[1:n], type = "l", col = "red", ylim = c(0,200),
     ylab = "steps",
     xlab = "amount of observations taken into account")
lines(seq(1,n), a_2[1:n], type = "l", col = "orange")
lines(seq(1,n), a_3[1:n], type = "l", col = "blue")
```

```
lines(seq(1,n), a_4[1:n], type = "l", col = "dodgerblue")
lines(seq(1,n), a_5[1:n], type = "l", col = "green")
lines(seq(1,n), a_6[1:n], type = "l", col = "greenyellow")
legend("topleft", legend=c("hc, BIC", "hc, BDe","gs, mi",
                           "gs, x2","iamb, mi","iamb, x2"),
       col=c("red", "orange","blue","dodgerblue",
             "green","greenyellow"),
       lty=1, cex=0.8)
```

References

A.M. Hanea D. Kurowicka, R.M. Cooke and D.A. Ababei (Mar. 2010). 'Mining and visualising ordinal data with non-parametric continuous BBNs'. In: *ComputationalStatisticsandDataAnalysis* 54, pp. 668–687 (cit. on p. 44).

Ababei, Dan (2008). *UNINET help*. URL: `https://www.lighttwist-software.com/wp-content/doc/UninetHelp.pdf` (cit. on p. 43).

Agresti, A. (2013). *Categorical data analysis*. 3rd ed. Wiley (cit. on p. 21).

Bhatta, Sanjheev (May 2017). 'Conditional correlation analysis'. MA thesis. Tribhuvan University (cit. on p. 39).

David Heckerman, Dan Geiser and David M. Chickering (Sept. 1995). 'Learning Bayesian Networks: The Combination of Knowledge and Statistical Data'. In: *Machine Learning*, pp. 197–243 (cit. on p. 19).

Edwards, David (2000). *Introduction to Graphical Modelling*. 2nd ed. Vol. 20. Springer. ISBN: 978-1-4612-0493-0 (cit. on pp. 21, 67).

Fetsje Bijma, Marianne Jonker and Aad van der Vaart (2016). *An introduction to mathematical statistics*. Amsterdam university press. ISBN: 978-90-5041-135-6 (cit. on p. 32).

Flaminia Musella, Paola Vicard and Vincenzina Vitale (Sept. 2019). *Copula Grow-Shrink Algorithm for Structural Learning*. Springer and Cham. ISBN: 978-3-030-21140-0 (cit. on p. 27).

Ioannis Tsamardinos, Constantin F. Aliferis and Alexander Statnikov (Jan. 2003). 'Algorithms for Large Scale Markov Blanket Discovery'. In: *Department of Biomedical Informatics* (cit. on p. 28).

John, George H. and Pat Langley (July 1995). 'Estimating Continuous Distributions in Bayesian Classifiers'. In: *Stanford University*, pp. 338–345 (cit. on p. 66).

Lasso vs Ridge vs Elastic Net | ML (Mar. 2020). `https://www.geeksforgeeks.org/lasso-vs-ridge-vs-elastic-net-ml/`. pseudonym: pawangfg (cit. on p. 38).

Marco Scutari, Catharina Elisabeth Graafland and Jose Manuel Gutiérrez (2018). 'Who Learns Better Bayesian Network Structures: Constraint-Based, Score-based or Hybrid Algorithms?' In: *Proceedings of Machine Learning Research* 72, pp. 416–427 (cit. on pp. 65, 67).

Margaritis, Dimitris (May 2003). 'Learning Bayesian network model structure from data'. PhD thesis. Carnegie Mellon University (cit. on p. 27).

Pearl, Judea (Dec. 1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann. ISBN: 978-0-934613-73-6 (cit. on pp. i, 16).

Qshick (Jan. 2019). *Ridge Regression for Better Usage*. URL: `https://towardsdatascience.com/ridge-regression-for-better-usage-2f19b3a202db` (cit. on p. 36).

Scutari, Marco (Jan. 2020a). *Cross-validation for Bayesian networks*. URL: `https://www.bnlearn.com/documentation/man/bn.cv.html` (cit. on p. 45).

– (Jan. 2020b). *Measure arc strength*. URL: `https://www.bnlearn.com/documentation/man/bn.cv.html` (cit. on p. 46).

– (Jan. 2020c). *Score functions: computing & comparing*. URL: `https://www.bnlearn.com/examples/score/` (cit. on p. 46).

Scutari, Marco and Jean-Baptiste Denis (May 2014). *Bayesian Networks with Examples in R*. Chapman and Hall. ISBN: 978-1-4822-2558-7 (cit. on pp. i, 4, 11, 15, 22, 26, 32, 35, 38–40).

Sucheta Nadkarni, Prakash P. Shenoy (Sept. 1998). 'A Bayesian network approach to making inferences in causal maps'. In: *European Journal of Operational Research* 128, pp. 479–498 (cit. on p. 9).

Tsamardinos, Ioannis and Giorgos Borboudakis (2010). 'Permutation testing improves Bayesian network learning'. In: *Machine Learning and Knowledge Discovery in Databases*, pp. 322–327 (cit. on pp. 21, 67).

unknown (2019). *A comparison of the Pearson and Spearman correlation methods*. Minitab, LLC. URL: `https://towardsdatascience.com/ridge-regression-for-better-usage-2f19b3a202db` (cit. on p. 40).