Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia
COPPE UFRJ

BAYESIAN NETWORK STRUCTURE LOCAL LEARNING FROM INCOMPLETE DATA

Roosevelt de Lima Sardinha

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Gerson Zaverucha

Rio de Janeiro
Fevereiro de 2014

BAYESIAN NETWORK STRUCTURE LOCAL LEARNING FROM INCOMPLETE DATA

Roosevelt de Lima Sardinha

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2014

Sardinha, Roosevelt de Lima

Bayesian Network Structure Local Learning from Incomplete Data/Roosevelt de Lima Sardinha. – Rio de Janeiro: UFRJ/COPPE, 2014.

XII, 91 p.: il.; 29, 7cm.

Orientador: Gerson Zaverucha

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2014.

Bibliography: p. 80 – 84.

1. bayesian networks. 2. local learning. 3. incomplete data. I. Zaverucha, Gerson. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

APRENDIZADO LOCAL DE ESTRUTURAS DE REDES BAYESIANAS COM DADOS INCOMPLETOS

Roosevelt de Lima Sardinha

Fevereiro/2014

Orientador: Gerson Zaverucha

Programa: Engenharia de Sistemas e Computação

Neste trabalho, o BBSL (Bayes Ball Structure Learning), um algoritmo para aprendizado de estrutura de redes bayesianas a partir de dados incompletos é apresentado. O BBSL melhora a classificação de uma variável escolhida pelo usuário alterando a estrutura da rede nos arredores da variável de classe. Esse algoritmo é capaz de lidar com dados incompletos utilizando o critério de d-separação, adotado no algoritmo Bayes Ball, uma alternativa ao uso da Markov Blanket para dados incompletos. BBSL é também um algoritmo de aprendizado local, o que permite que sua execução não seja dependente do número de variáveis do domínio consideradas, sendo uma alternativa de execução rápida a algoritmos presentes na literatura.

Os resultados obtidos neste trabalho mostram que o BBSL é uma alternativa eficiente para o aprendizado que visa melhorar a classificação de uma dada variável. O BBSL é comparado com um algoritmo baseado em restrições (GS), um híbrido (MMHC) e um baseado em pontuação (SEM com GHC), mostrando bons resultados em termos de tempo, comparável a opção baseada em restrições, e em termos de score CLL (Conditional Log-Likelihood). A abordagem local, usando d-separação, e score próprio para problemas de classificação, contribui para que a busca seja mais eficiente ao restringí-la a um determinado conjunto de redes bayesianas com modificações locais, reduzindo o tempo de execução do algoritmo, mesmo com dados incompletos.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

BAYESIAN NETWORK STRUCTURE LOCAL LEARNING FROM INCOMPLETE DATA

Roosevelt de Lima Sardinha

February/2014

Advisor: Gerson Zaverucha

Department: Systems Engineering and Computer Science

In this work, BBSL (Bayes Ball Structure Learning), an algorithm for learning the structure of bayesian networks from incomplete data is presented. BBSL improves the classification of a variable chosen by the user by changing the network structure in the surroundings of the class variable. This algorithm is able to work with incomplete data using d-separation criteria, in Bayes Ball, an alternative to the use of the Markov Blanket for incomplete data. BBSL is also a local learning algorithm, what allows it to execute independently of the number of variables in the domain considered, being a fast execution alternative to algorithms present in the literature.

The results obtained in this work show that BBSL is an efficient alternative to learning for enhancing the classification of a specific variable chosen by the user. BBSL is compared to a constraint-based algorithm (GS), a hybrid one (MMHC) and a score-based one (SEM with GHC), presenting good results in terms of time, comparable to the constraint-based option, and in terms of CLL (Conditional Log-Likelihood) score. The local approach, using d-separation, and appropriate score to classification tasks, contribute to a more efficient search by restricting it to a specific set of bayesian networks with local changes, reducing the algorithm's execution time, even in the presence of incomplete data.

Contents

List of Figures

List of Tables

Abbreviations

BD	Bayesian Dirichlet
CLL	Conditional Log-Likelihood
DMBC	Dynamic Markov Blanket Classifier
BB	Bayes Ball
ALARM	A Logical Alarm Reduction Mechanism
MMPC	Max-Min Parents and Children
BARD	Bayesian Aerosol Release Detector
GHC	Greedy Hill Climbing
ESS	Expected Sufficient Statistics
CPT	Conditional Probability Table
MIT	Mutual Information Tests
NML	Normalized Maximum Likelihood
AIC	Akaike Information Criterion
GES	Greedy Equivalent Search
BBSL	Bayes Ball Structure Learning
DAG	Directed Acyclic Graph
MCMC	Monte Carlo Markov Chain
MMHC	Max-Min Hill Climbing
MDL	Minimum Description Length
LL	Log-Likelihood
RAM	Random Access Memory
BIC	Bayesian Information Criterion
SEM	Structural Expectation Maximization
BN	Bayesian Network
EM	Expectation Maximization

GS	Grow and Shrink
TLSA	Two-Level Simulated Annealing
SLL	Score-based Local Learning
SHD	Strcutural Hamming Distance
GB	Gigabyte
GSMB	Grow and Shrink Markov Blanket
DAHVI	Discriminative Approach for Hidden Variable Intro- duction
IAMB	Incremental Association Markov Blanket
MB	Markov Blanket
SS	Sufficient Statistics
MI	Mutual Information
ITS	Intelligent Tutoring Systems
KS	Koller-Sahami

Chapter 1

Introduction

In this chapter, an introductory view of Bayesian networks is presented. Also, the motivations that led us to develop this research, and to obtain a local structure learning Bayesian network algorithm that, as shown in the presented results, is able to improve the classification of a variable more than commonly used algorithms in the literature. Applications of Bayesian networks are as well presented to support our motivations, like medical diagnosis, genetics and reliability analysis. Moreover, the objectives of this work and the topics exhibited in the next chapters are specified.

A Bayesian Network is a probabilistic graphical model able to represent, in a compact way, a joint distribution of variables [1]. Besides this, it is able to answer questions about this distribution, by the use of inference algorithms without obtaining the joint distribution of all variables, in a more elegant, and computationally less expensive way. Bayesian Networks can also be applied to a great variety of classification and prediction problems that can be modeled as this kind of network, such as genetic linkage analysis, speech recognition, information theory and reliability analysis [2]. As such, they have been widely used for uncertain reasoning in artificial intelligence and expert systems [1]. Several applications in classification problems have been developed like in diagnosis, in troubleshooting, in data mining and in pattern recognition. Bayesian networks, unlike other methods, are able to represent the relationship between variables in a human-readable way, through local probability distributions, conditional independence and independence representations in a directed acyclic graph. They are also able to deal with incomplete data preserving important characteristics from the missing data, like the missing data distribution

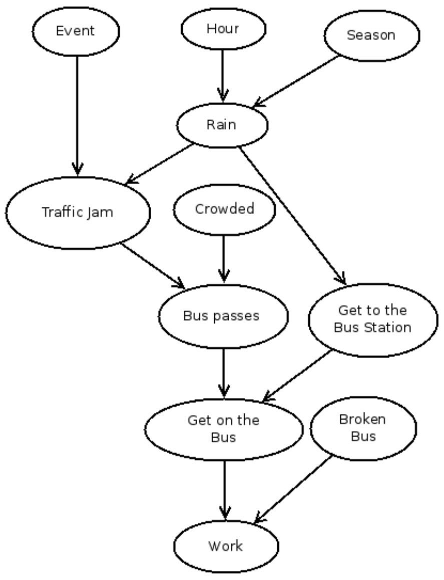


Figure 1.1: Bayesian Network structure example

pattern.

From a Bayesian network representation of a problem like the one shown in Figure 1.1, one is able to answer queries involving the probability of a variable assuming a certain value, given some evidence about the values of the others in the variable set. This way queries can be answered, for instance: what is the probability of variable **Rain** to assume value **little**, given that variable **Season** has assumed value **summer** and variable **Traffic Jam** has assumed value **low**? These queries can involve any set of variables in the network. But not only this kind of question but other useful ones can be answered, like: Which one is the most probable value to variable **Get to the Bus Station** to assume, since variables **Rain**, **Crowded** and **Bus passes** have assumed values **little**, **much** and **yes**, respectively? And even more immediate ones that can arise from the direct interpretation of the graph.

When using a Bayesian network we count on a set of tools already developed by the scientific community in the form of exact and approximate algorithms, for tasks such as inference and learning. As examples of inference algorithms found in the literature, and some of them have already some software implementation available, there are *Variable Elimination* [3], *Junction Tree*[3] and approximated methods MCMC (Markov Chain Monte Carlo) based like *Gibbs Sampling* [3]. For learning parameters of a Bayesian Network we have Gradient Ascent [3] and Expectation Maximization [3]. And finally, for learning the structure of a Bayesian network from a given dataset there are *Grow and Shrink* [4], *Optimal Reinsertion* [5] , *Sparse Candidate* [6], *Max-Min Hill Climbing* [7], *Greedy Equivalent Search* [8], *Simulated Annealing* [9], *Dynamic Markov Blanket Classifier* [10], B&B[11], and others.

1.1 Motivation

Bayesian networks, as shown in the book "Bayesian networks: a practical guide to applications" [12], have several interesting applications. Here some of them are exemplified:

- Medical Diagnosis - since medical decisions are hard to do, and human-decision making is not optimal for complex problems, under bad conditions, and with plenty of information, physicians need assistance on decision-making and risk evaluations. Bayesian networks can be applied to help in diagnosis, prognosis, and selection of therapy. Other applications are in clinical epidemiology, disease surveillance, BARD (Bayesian Aerosol Release Detector), prediction of secondary structure of a protein and discover causal pathways in gene expression data, the ALARM network for monitoring patients in intensive care, diagnosing oesophageal cancer, mammography, diagnosing liver disorder and assessment of cardiovascular risk;

- Genetics - discovering the genetic base for diseases and traits is a difficult work, mainly when considering several possible genotypes and interaction from these genotypes with the environment. Machine learning methods have been applied to this problem and BNs are useful to discover genetic basis of complex traits on the same time it provides prognostic models;

- Biomedicine and health-care;

- Decision support on crime risk factors in an area, helping analizing crime patterns. It has been adopted as a useful approach for inference and prediction in the crime problem domain. Also for forensic science, when dealing with multiple sources of uncertainty;

- Classifying potential for mineral deposit occurrence by using a Bayesian classifier to determine from some known barren or mineralized area, how mineralized are the new areas being evaluated;

- Student modeling for use in Intelligent Tutoring Systems (ITS);
- Sensor validation algorithms - to evaluate if the output of a sensor indicates a real value or if its just a sensor fault, avoiding wrong behaviour from sensor dependent systems;
- Information retrieval;
- Reliability analysis of safety-critical systems;
- Terrorism risk management;
- Financial applications - as credit-rating of companies (how desirable is a company for investments), usually determined by specialists. BNs allow missing data, and allow better evaluation than other methods, by making it possible to evaluate the interactions between variables that caused the results.

Other applications cited at this book include: classification of wines, pavement and bridge management, complex industrial processes operation, risk management in robotics and human cognition enhancement. After all, applications presented usually work with decision support, reliability analysis, complex tasks for a human to reason about, and tasks involving uncertainty and possibly missing data.

In [13], Bayesian Networks are also said to be useful to evaluate the impact of management maturity level, assess the usability of web sites using web logs, monitoring bioreactors and to analyze customer survey data.

Although Bayesian networks are very useful, as shown in the last paragraphs, it is necessary to build its structure and to determine its parameters before any inference can be made. For this task, a commonly used approach is to build the BN from the knowledge of a specialist and from the available literature. However, this task is usually known as difficult and time consuming [14]. Furthermore, it may lead to imprecise networks, since it often depends on the opinion of the specialists involved. As shown in [14] it is still a useful aproach when data is unavailable.

Another approach, is to learn the network structure and parameters from data, avoiding the difficulties presented before. It brings advantages like freeing the specialist time for use in other tasks. This is the approach used in this work.

Real-world applications usually have missing data, frequently as a result of the process of gathering it. Sometimes data is difficult to obtain, the process of gathering it has failures, or the missing value is a consequence of the behaviour of other variables involved. As cited in [15], in a drug study a missing datum may be caused by a patient that abandoned the research due to the side effects of it, becoming too sick to continue.

Actually, several Bayesian network structure learning algorithms require complete data for learning, what is incompatible with the situation of having missing data in real-world applications. For instance, the constraint-based category of algorithms, and popular score-based solutions like GHC (Greedy Hill Climbing) [3], K2 [16], Simmulated Annealing [9] need complete data to run. Even hybrid solutions like MMHC [7] and H2PC [17], which start with a constraint-based approach, also require no missing data. Alternatives to work with incomplete data, that consider the data incompletion pattern, are restricted, and include algorithms like SEM (Structural Expectation Maximization) [18] and some SEM-based variants for structure learning. However, because of the high number of inferences demanded by working with incomplete data, these approaches still take too much computation time.

In order to reduce the time consumed by the global SEM-based approaches, it is proposed in this work a local alternative. The local approach for Bayesian network structure learning is already present in algorithms like SLL [19] and DMBC [10], but for complete data. For complete data, as in SLL, the local approach has shown to be very fast. In [19] it is shown that a score-based local approach can have an execution time competitive to constraint-based solutions. For incomplete data, as far as we know, only DAHVI (Discriminative Approach for Hidden Variable Introduction) [20], a structure revision algorithm that tries to improve the classification of a variable by introducing hidden variables applies it.

United to all that was cited, several of the problems listed before where Bayesian networks can be used are machine learning classification problems. Then, improving the classification of a variable or group of variables is a desirable feature of an algorithm for Bayesian networks, as proposed in this work.

The approach proposed uses the d-separation concept, that will be explained later. It allows us to determine which are the variables that influence the classification of a particular one in the algorithm to be presented. Since it may be used with incomplete data, selecting variables accordingly to which ones are in the evidence, it consists of a more flexible approach then the Markov Blanket one. The MB of a variable consists of the minimum set of variables conditioned on which a variable is conditionally independent of all others in the directed acyclic graph. It is a set composed by the parents, children and spouses of the variable being evaluated. Since it assumes these variables are in the evidence ("conditioned on which"), in the case of unobserved variables, where these variables may not be in the evidence, it does not apply without adaptations. The d-separation concept works in this situation, being able to deal with incomplete data, where not all MB variables are in the evidence. In addition, an algorithm like Bayes Ball is able to return the set of variables that influence the probability distribution of the class variable using d-separation criteria.

1.2 Objectives

Based on the motivations exposed before, the objectives of this work include:

- To build an algorithm that can be applied to improve the classification of a variable in the Bayesian network, through Bayesian network structure learning;
- To learn from incomplete data, and to allow previous knowledge usage about the domain;
- To accomplish the learning task as fast as possible, with good results, concerning the adequated score;
- To show that d-separation criteria can be used in the learning task, as an alternative to the Markov Blanket set.

1.3 Organization

The following chapters are organized as follows:

- Chapter 2 briefly introduces the knowledge needed to understand the rest of the work. Extended explanations can be taken in the referred articles.

This chapter does not intend to be an exaustive exposition of the knowledge available in the respective approached topics;

- Chapter 3 explains the algorithm proposed in this work, and the score used to evaluate the networks;
- Chapter 4 is a review of the available Bayesian network structure learning literature, related works are presented there;
- Chapter 5 are the results obtained and the discussion about the presented results;
- Chapter 6 presents conclusions.

Chapter 2

Background Knowledge

The background knowledge chapter was developed to introduce to the reader the key concepts that will be used in the presentation of this work. In order to achieve this, a more formal presentation of Bayesian networks is done, as well as, other concepts like conditional independence in this probabilistic graphical model, d-separation and Bayes Ball algorithm, structure learning and useful algorithms like EM and SEM to work with incomplete data, and finally the scores used, are presented.

2.1 Bayesian Networks

A Bayesian network is a directed acyclic graph (DAG) where nodes represent random variables that belong to the variable set of the domain of a problem, and each edge represent a statistical relationship of dependence between two variables. To each variable is associated a conditional probability table (CPT). The CPT denotes a local probability distribution, from the variable represented by the node given its parents in the graph [1].

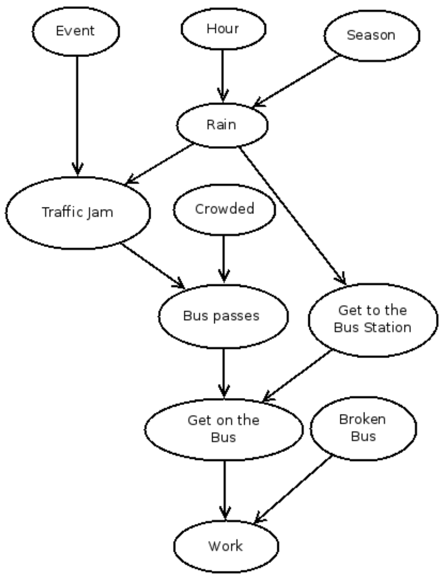


Figure 2.1: Bayesian Network structure example

This way, in a Bayesian network, one is able to represent the conditional independence relationships existent among variables, the independence relationships, and the joint distribution of all variables in the graph. In a Bayesian network, a node or set of nodes is considered independent of one other node or set of nodes if they are disconnected in the graph, that is, if there is no undirected path from one set of nodes to another. Also, a variable is conditionally independent of all its nondescendants given its parents in the DAG.

Other characteristic of Bayesian networks, as it will be discussed later, is the assumption that a variable is conditionally independent of all other variables in the network given its Markov Blanket (MB) [21]. Besides all this, a variable is independent of all other variables, as it also will be discussed later, given its Bayes Ball (BB) set of variables, without the restriction that all the MB variables are in the evidence [22].

Furthermore, the joint distribution of all variables cited previously can be obtained by computing the product of the probabilities of each variable given its parents, as in the Equation 2.1 [1]:

$$P(X_1 \dots X_n) = \prod_i P(X_i|\mathbf{Parents}(X_i)). \tag{2.1}$$

Where $\mathbf{Parents}(X_i)$ means the set of parents of the variable X_i in the graph,

Table 2.1: Example of CPT for three variables X, Y and Z

X	Y	Z	
yes	yes	yes	0.1
yes	yes	no	0.2
yes	no	yes	0.1
yes	no	no	0.1
no	yes	yes	0.9
no	yes	no	0.8
no	no	yes	0.9
no	no	no	0.9

if the variable has no parents the set is $\emptyset$. To compute the joint distribution it is necessary to compute the formula for every combination of values of X_1 to X_n, what is unfeasible in networks with great number of parameters.

Given a topological order of the variables in the network $(X_1, ..., X_n)$, Equation 2.1 can be obtained by the application of the restriction imposed by Bayesian networks of each variable being conditionally independent of its nondescendants given its parents to the chain rule of probability, as shown below.

Accordingly to the chain rule, we have:

$$P(X_1, \ldots, X_n) = \prod_i P(X_i | X_1, \ldots, X_{i-1}). \tag{2.2}$$

Applying the cited conditional independence rule for Bayesian networks, we get back to Equation 2.1.

Associated to each variable in the Bayesian network is a Conditional Probability Distribution of the variable given its parents in the graph, as said before. When variables are discrete a Conditional Probability Table (CPT) is built with all possible combinations of values of the variable and its parents. Assigned to each combination of values in this table is a probability value, called the *parameters* of the network.

On Table 2.1, Y and Z are parents of X and each of the three variables has only two possible values, **yes** or **no**. The probabilities in each line are the parameters associated to the values on the same line. That is, the probability of X given Y and Z, considering the values in the line.

2.2 Conditional Independence in Bayesian Networks

In a Bayesian network if there is an edge from one node to another, it means that the second node is statistically dependent on the first. So, if the probability distribution of the first node is changed, there may be some impact on the probability distribution of the second. That does not necessarily mean that the first variable is the cause of the second, unless you are dealing with a causal network [1].

A causal network is a network where the dependency between the variables are not only statistical but also causal [1]. What is meant is that, in this kind of network, if there is a connection between two variables, from **A** to **B**, then we have that **A** causes **B**, or that **B** is a consequence of **A**.

From the definition of Bayesian networks, all nodes that are not descendants of the second variable (**B**), are conditionally independent of the variable (**B**), given its set of parents, to which the first node (**A**) pertains to. Explaining, in Figure 2.1, **Bus passes** is conditionally independent of its nondescendants (**Event**, **Rain**, **Hour**, **Season**, **Get to the Bus Station**, **Broken Bus**), given that its parents **Traffic Jam** and **Crowded** are in the evidence of the query. It means that, the intensity of the **Rain** (the value of the variable **Rain**), does not affect that the bus is passing or not by the bus station, if we already know that the bus is crowded and that there is no traffic jam. In a mathematical form:

$$\begin{aligned} P(Bus\ passes|Crowded, Traffic\ jam, Rain) = \\ P(Bus\ passes|Crowded, Traffic\ jam). \end{aligned} \tag{2.3}$$

But what about the descendants of variable **Bus passes**? Does the value observed in the variable **Work** affects the distribution of variable **Bus passes**? The answer to this question depends on if we already know or do not about the variable **Get on the bus**. If **Get on the bus** is not in the evidence, then the value of **Work** gives us some evidence about the value of **Get on the bus** and so affects the distribution of the variable **Bus passes**. This kind of behaviour is the object of studies in this section, and can be explained by the use of the concept of *D-separation*.

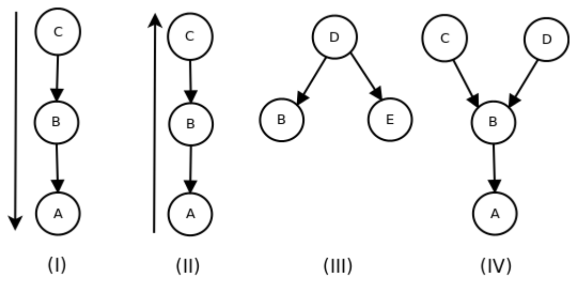


Figure 2.2: D-separation cases

D-separation as defined in [3]:

Definition 2.2.1 *Let* $\mathbf{X}$*,* $\mathbf{Y}$ *and* $\mathbf{Z}$ *be three sets of nodes in a graph* G*. We say that* $\mathbf{X}$ *and* $\mathbf{Y}$ *are* d-separated *given* $\mathbf{Z}$*, denoted* $d_{sep_G}(\mathbf{X};\mathbf{Y}|\mathbf{Z})$*, if there is no active trail between any node* $X \in \mathbf{X}$ *and* $Y \in \mathbf{Y}$ *given* $\mathbf{Z}$ *[3].*

And also, still as said in [3], there is an active trail between two nodes whenever we have a *v-structure* ($X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$), and X_i or one of its descendants are in $\mathbf{Z}$, or, if no other node along the trail is in $\mathbf{Z}$.

For a better understanding of the d-separation criterion it is useful to separate the definition in four cases, illustrated in Figure 2.2

In the first case illustrated in Figure 2.2, **I**, the probabilistic influence of node C in node A only happens when node B is not in the evidence. When it is, there is no need to know about C distribution when computing the probability of A given the other nodes, because A is conditionally independent of C given B. Analogously, in Figure 2.1, if you know that there is a huge **Traffic jam**, it is not probable that your bus will pass (variable **Bus passes**), no matter the size of the **Event** that is happening in the city. However, if you do not know about the **Traffic jam**, the size of the event can give you a clue about it, and so let you change the probability of **Bus passes**.

In the second case, **II**, something similar occurs in the opposite direction, the distribution of A only affects the one of C if B is not in the evidence, if it is then you do not have to worry about the value of A, when computing the probabilities for C. Following the same analogy, if you know that **Bus passes** probably there is not a big **Trafficjam**, and so, probably there is not a big **Event** happening in the city, so when you do not know about the **Traffic Jam**, **Bus passes** affects the distribution of **Event**. Otherwise, if it is already known about the intensity of the **Traffic jam**,

Table 2.2: CPTs for B, D and E

B	D	
yes	yes	0.6
yes	no	0.8
no	yes	0.4
no	no	0.2

E	D	
yes	yes	0.3
yes	no	0.6
no	yes	0.7
no	no	0.4

D	
yes	0.4
no	0.6

Table 2.3: Probabilities of B given D and E, B given E, B given D, and probability of B

B	D	E	
yes	yes	yes	0.6
yes	yes	no	0.6
yes	no	yes	0.8
yes	no	no	0.8
no	yes	yes	0.4
no	yes	no	0.4
no	no	yes	0.2
no	no	no	0.2

B	E	
yes	yes	0.75
yes	no	0.69
no	yes	0.25
no	no	0.31

B	D	
yes	yes	0.6
yes	no	0.8
no	yes	0.4
no	no	0.2

B	
yes	0.72
no	0.28

this affects the distribution of **Event** and not **Bus passes** anymore. There is no need to know about the value assumed by **Bus passes** in this last situation.

In the third case, **III**, following a little bit different directives from that taken to explain the other two cases, consider the following CPTs, on Table 2.2.

Computing the probabilities $P(B|D, E)$, $P(B|E)$, $P(B|D)$, and $P(B)$ it results in Table 2.3.

The values on Table 2.3 were calculated based on the values of CPTs adopted for nodes in Figure 2.2 presented on Table 2.2. Notice that for this specific case, randomly generated, when D is present in the evidence, the value of E does not affect the probability of B, as it can be seen by the same values in the tables where are calculated $P(B|D, E)$ and $P(B|D)$. However if we do not have D in the evidence, the values now are different for $P(B|E)$ and also for $P(B)$. This gives us some evidence about this behaviour, and that is what in fact happens, when D is in the evidence it blocks the probabilistic influence of E on B, and of B on E. In other words B and E are conditionally independent given D.

The fourth case **IV** is probably the most difficult to understand and the most intriguing one. In this situation, shown in Figure 2.3, when variable B or any of its descendants is in the evidence, the value of variable C can affect the distribution of variable D, and vice-versa. Only when none of the variables including B and its descendants are in the evidence, the path between variables C and D will be blocked, such that it is not an active trail anymore. As a way of illustrating this behaviour, consider the corresponding structure in the Bayesian network in Figure 2.1 that includes nodes **Traffic jam**, **Rain**, **Event**, **Bus passes**, **Get on the bus**. Suppose the value of **Traffic jam** is known, for example, and that it assumes value big, meaning that there is a big traffic jam. If it is known that a very big **Event** is happening, the possibility of **Rain** is changed, since it is known about the **Traffic jam**. The same way, if we know that it is not raining, there is a greater probability of having some big event in the city since it was noticed that there is a traffic jam. However, if the value of **Traffic jam** is not known, but one of its descendants is in the evidence, they give some hints about the probability of **Traffic jam** to assume its possible values, and this way the values of **Event** and **Rain** can affect the distribution of each other.

Combining the four situations cited before, and considering the variables that are and that are not in the evidence, it is possible to determine which variables in the Bayesian network can be influenced by a selected variable. It means that if the value of this variable is changed it is possible to determine which variables may have its distribution affected. So, considering this, and as an example, for variables **Get to the bus station** and **Event** being independent, there should not exist any active trail from one to another. There are two possible trails to check if they are active: a) the one that includes nodes **Event**, **Traffic jam**, **Rain** and **Get to the bus station**; and b) the trail that includes nodes **Event**, **Traffic jam**, **Bus passes**, **Get on the bus** and **Get to the bus station**. For the first, following the cases that were explained before, if variable **Rain** is not observed, and, variable **Traffic jam** or any of its descendants is observed, this path is active and so **Get to the bus station** and **Event** are not independent. But if these conditions are not satisfied the path is blocked and it is also necessary to evaluate the other trail, to see if it is active, if both are blocked than the variables are independent. Analyzing the second path, variable **Get on the bus** or any of its descendants should be in the evidence, and also variables **Traffic jam** and **Bus passes** should not have been observed. This way the path is active, otherwise it is blocked and not active. As said before, if any path is active between the variables then they are not independent.

The minimum set of variables that influence the value of one variable blocking influence from others is called the Markov Blanket of the variable, and it is composed by the parents of the variable, its children, and the parents of its children (also called its spouses). If some of these variables are not in the evidence, influence from other variables can be detected using the rules already presented. An algorithm that can be used to determine which variables influence a selected one, in the case where not all variables in the Markov Blanket are in the evidence, is the Bayes Ball algorithm [22].

2.3 Bayes Ball

Using d-separation allows us to determine which are the variables in a Bayesian network that affect the probability distribution of one specific variable even if not all variables are in the evidence, that means, for one specific example it is possible to determine the influence in the graph of one variable into another even if not all data is observed. In this context it is very immediate to think about an algorithm that could be able to return all the variables that may influence the probability distribution of a specific node, given some evidence. Algorithms like these exist and the one used in this work is called Bayes Ball. In [3], in section 3.3.3, an algorithm to find nodes reachable from X to Z via active trails is presented, as an example.

Bayes Ball, "the rational pastime", as described in [22], uses the d-separation concept to navigate through the Bayesian network. It starts from the query variable and walks through the graph while determining which are the variables that can change the probability distribution of a specific variable. The reason the use of the algorithm is needed, is related to the d-separation concept, because in case a variable is not present in the evidence, the Markov Blanket of the variable may not be the set of variables that is being looked for. For instance, if the parent of variable X is Y, and the latter was not observed, variable Z that is parent of Y can influence the distribution of X and is not in the Markov Blanket set of X. Situations like these agree with the d-separation criteria, previously explained, for the case where the dataset is not complete.

The Bayes Ball algorithm is presented in Algorithm 1.

Algorithm 1 Bayes Ball (adapted from [22])

```
function BayesBall($J, K, bn\_structure$)
    $visited$, $marked\_on\_top$, $marked\_on\_bottom = \emptyset$
    $visited$, $marked\_on\_top$, $marked\_on\_bottom = \emptyset$
    $nodes\_to\_be\_visited = J$
    Mark all nodes as visited from children
    while $nodes\_to\_be\_visited \neq \emptyset$ do
        $j$ = get and remove node from $nodes\_to\_be\_visited$
        $visited$.append($j$)
        if ($j$ not in $K$) and ($j$ is visited from a child) then
            if (top of $j$ not marked) then
                $marked\_on\_top$.append($j$)
                $nodes\_to\_be\_visited$.append(parents($j$))
            end if
            if ($j$ bottom is not marked) then
                $marked\_on\_bottom$.append($j$)
                $nodes\_to\_be\_visited$.append(children($j$))
            end if
        end if
        if ($j$ is visited from a parent) then
            if ($j$ in $K$) and (top of $j$ not marked) then
                $marked\_on\_top$.append($j$)
                $nodes\_to\_be\_visited$.append(parents($j$))
            end if
            if ($j$ not in $K$) and ($j$ bottom is not marked) then
                $marked\_on\_bottom$.append($j$)
                $nodes\_to\_be\_visited$.append(children($j$))
            end if
        end if
    end while
                    ▷ The nodes that affect $J$ given $K$ are those nodes in $visited$.
    return $visited$
end function
```

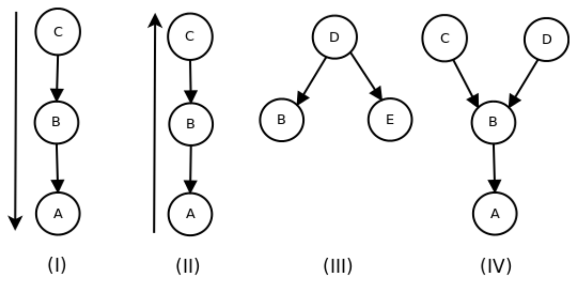


Figure 2.3: D-separation cases

To understand this algorithm it can be useful to think about the four cases of d-separation presented in the beginning.

Lets divide our strategy into two parts, one concerning visiting the node from a child and the other concerning visiting the node from a parent.

Consider the following paths:

I) In Figure 2.3 (I), from node C to A visiting node B from parent;

II) In Figure 2.3 (II), from node A to C visiting node B from child;

III) In Figure 2.3 (III), from node B to E visiting node D from child;

IV) In Figure 2.3 (IV), from node C to D visiting node B from parent.

If the node is visited from a child, there are two possibilities: a) the variable is in the evidence; or b) the variable is not in the evidence. Situation I and IV does not apply because the visit is from the parent. In the first case (a), analyzing situation II, the path is blocked. For situation III the path will also be blocked. So for all four cases, there is nothing to do, the path is not active and the visit would return to the child node. In the second case (b), for situation (II) the path is not blocked and the parents should be visited. For situation (III), the children should be visited. So if the variable is in the evidence (a), the algorithm should do nothing and if the variable is not in the evidence (b) parents and children of the variable should be visited. It explains the first condition inside the loop.

Analogously for the case where the visit is from a parent, there is also the same two possibilities called previously (a) and (b). In this situation only I and IV apply. In the first case (a), the path is blocked in situation (I), however, parents should be visited in situation (IV). In situation (b), children should be visited for situation (I) and the path (IV) is blocked. And now, these explain the second condition inside the loop, and its inner conditions.

Notice that for the d-separation case illustrated in Figure 2.3 (IV), if A was in the evidence the flow should not be interrupted. And it is not, since the algorithm would visit B and A, and return to visit D. However, if any of the descendants of B is in the evidence (A in this case), the algorithm would visit B and A, but this time it would not return to visit D.

2.4 Structure Learning in Bayesian Networks

Bayesian network structure learning tries to solve the problem of determining the structure of a Bayesian network given data. Considering that the variables are known, it learns the relationships existent among them. This is represented in the graph by the edges that connect variables and its orientation. Besides this, it learns the independence and conditional independence constraints that exist in the distribution, since the structure of the graph makes it possible to read which variables are independent or conditionally independent of each other given another set of variables, as explained in the last section. The structure with its parameters is also a compact representation of the joint probability distribution of the variables, such that it could also be obtained. However, it is not usually desirable since you can make inference without obtaining the joint probability distribution, and computing it can be computationally expensive.

There are three main approaches for learning the structure of Bayesian networks: the score-based, the constraint-based and the hybrid one. The last is a mixture of characteristics of the first two in the same algorithm. Each of them will be explained in this section.

2.4.1 Score-Based Structure Learning

Learning the structure of a Bayesian network using a score-based approach demands the definition of a score function to allow structure evaluation. This score function will be responsible for determining whether a structure is better than another or not, or how well does this structure fits the data. There are several kinds of possible scores and this subject is left for a particular section in this work. In summary, the score is a function that returns a value of how good is the structure for a particular

Table 2.4: Number of possible Bayesian networks given the number of variables

Number of variables	Number of possible DAGS
1	1
2	3
3	25
4	543
5	29,281
6	3,781,503
7	1,138,779,265
8	78,370,2329,343
9	1,213,442,454,842,881
10	4,175,098,976,430,598,100

purpose, and should be evaluated in order to make comparisons among networks.

To determine the best network structure accordingly to a given score it is necessary to search the space of all possible Bayesian networks. One simple approach would be to take each possible network, calculate the score for each one, and compare the results for all of them. Unfortunately, this approach is not efficient since the number of possible Bayesian networks increases very fast with the number of variables in the network. The computation of this number is possible by calculating the number of DAGs (Directed Acyclic Graphs) which is the same number of possible Bayesian networks since a Bayesian network is a DAG. There is a recursive formula for this computation [23] [24] and some of the assumed values were computed in Table 2.4 as a way of illustrating this super-exponential behaviour:

$$f(n) = \sum_{i=1}^{n} (-1)^{i+1} C_i^n 2^{i(n-i)} f(n-i). \tag{2.4}$$

Considering this fast increasing number of networks, another approach needs to be considered. For several practical problems it is not a requisite to have the best solution, it is only desirable, such that a good solution is usually enough. So, an heuristic is applied to search the space of possible Bayesian network structures, computing the score only for some of the networks. And, as well as in the case of the score, there are several possible known heuristics described by algorithms like K2 [16] or Greedy Hill Climbing [3].

As an example, it would be useful to describe one of them, Greedy Hill Climbing, in order to illustrate the presented ideas.

Greedy Hill Climbing

The pseudocode for the Greedy Hill Climbing algorithm is shown in Algorithm 2.

Algorithm 2 Greedy Hill Climbing

```
function GHC($initial\_bn, score\_function$)
    $current\_bn$ = $initial\_bn$
    $score\_improves$ = True
    while $score\_improves$ do
        $N$ = Compute neighbours for current Bayesian network
        $next\_eval$ = -infinity
        $next\_bn$ = null
        for $bn$ in $N$ do
            $eval\_bn$ = Evaluate Bayesian network using score function
            if $eval\_bn > next\_eval$ then
                $next\_bn = bn$
                $next\_eval = eval\_bn$
            end if
        end for
        $eval\_current\_bn$ = Evaluate $current\_bn$ using score function
        if $next\_eval \leq eval\_current\_bn$ then
            $score\_improves$ = False
        else
            $current\_bn = next\_bn$
        end if
    end while
    return $current\_bn$
end function
```

In the algorithm, from an existent starting structure that can be obtained by several means like by expert description, randomly, by running another algorithm and so on, all the possible neighbours of the starting network are generated and evaluated. At each iteration of the most internal loop these neighbours are evaluated and compared, if the score is improved by any of the neighbours then the algorithm continues to the next iteration of the outer loop, but if not, the algorithm ends and the current best network is returned. Neighbours, in this case, are networks that differ from the current structure from only one operator application. In the most common algorithm, there are only three possible operators: addition, removal or reversion of an edge. The number of possible neighbours of a network structure can be very big if the number of variables is too large. Such that the number of neighbour structures being evaluated at each iteration can still be big, and make the algorithm to consume too much time, when, for example, working with a large number of variables, examples and incomplete data.

2.4.2 Constraint-Based Learning

Constraint-based learning tries to learn the structure of the network by constructing a minimal I-map for the independence constraints in the domain [3]. A graph is an I-map for a determined set of independence constraints when all the independence constraints of this set are represented in the Bayesian network graph [3]. That is, the set of independence constraints in the set (for this case, in the domain), is a subset of the independence constraints represented by the graph. Continuing from this, a minimal I-map for a domain, is a graph where all the independence constraints are represented in the graph (it is an I-map), and if any edge is removed (and it does not matter which edge, and it can be just one) the graph is not an I-map anymore. This is the structural representation that constraint-based methods look for [3].

Independence Tests

In order to achieve the objectives described in the previous paragraph, one has to be able to test for independence between variables. This way, it is possible to determine which variables are connected to which other. Unfortunately, the problem of determining independence and conditional independence tests that are efficient for every dataset is not trivial. A common approach uses statistical tests to determine if the variables are independent or not, assuming as null hypothesis the case where the variables are independent.

The chi-squared statistic is able to compute a sum of differences between observed values and expected ones, and so is useful for this kind of application. In this case, the expected values being considered are the values for the case when the variables are independent. Another possible approach is using the mutual information statistic [25], since it calculates a ratio between the independent case and the dependent case. Both formulas need to be adapted to use sufficient statistics instead of the real probabilities, since they are unknown. As an illustration, these formula are shown:

$$\chi^2 = \sum_{i=1}^{n} (O_i - E_i)^2 / E_i; \tag{2.5}$$

$$\chi^2 = \sum_{x,y} (SS[x,y] - M \cdot \hat{P}(x) \cdot \hat{P}(y))^2 / (\hat{P}(x) \cdot \hat{P}(y)); \tag{2.6}$$

$$MI = \sum_{y \in Y} \sum_{x \in X} P(x,y) \log P(x,y) / (P(x) \cdot P(y)); \tag{2.7}$$

$$MI = \sum_{y \in Y} \sum_{x \in X} SS[x,y] \log SS[x,y] / (SS[x] \cdot SS[y]). \tag{2.8}$$

Equation 2.5 is the original formula for computing the Pearson's χ^2 statistic, where O_i are the observed values and E_i are the expected values for n examples. Adapting it to use the sufficient statistics Equation 2.6 is obtained. In this equation, X and Y are random variables of the Bayesian network, $SS[.]$ are the sufficient statistics obtained by counting the occurrences of the variables inside the brackets together, M is the total number of examples in the dataset and $\hat{P(x)}$ and $\hat{P(y)}$ are the estimated values for the probabilities of x and y respectively.

In the second pair of equations, analogously to the first pair, Equation 2.7 is the original formula for computing **mutual information** of two variables and Equation 2.8 is its adaptation using sufficient statistics. In χ^2 and MI modified formulas, when doing conditional independence tests, adaptations are needed to include the evidence set, originating slightly different formulas for χ^2 and MI. Other statistics in the statistical tests for independence are also possible like using G^2.

Heuristic

Constraint-based methods for learning network structure usually execute several independence and conditional independence tests until finding the best network representation for the distribution of the data. Each method considers a different heuristic in order to avoid the execution of too much independence tests, because testing all combinations of pair of variables given all combinations of possible variables in the evidence can take more time than desirable when the number of variables is big.

One important thing to notice is that by doing independence tests the only thing that is determined initially is the existence of the edge between variables, such that it is still necessary to orient these edges to achieve the full structure. Each algorithm has its specific heuristic for this step, but the observance of **v-structures** ($X \rightarrow Z \leftarrow Y$) and direction propagation of the edges is something often used for this task. Some algorithms also do not always give as output a network with all edges oriented, but a representation of an I-equivalence class, i.e., a set of Bayesian networks that represent the same set of independence constraints.

Grow and Shrink

As an example of constraint-based learning in this subsection, the Grow and Shrink (GS) Markov Blanket algorithm is presented. GS Markov Blanket (sometimes referred as GSMB) is used to recover the Markov Blanket of a variable [4]. It is used as part of the GS algorithm to recover a Bayesian structure, as presented in [4]. The algorithm is shown and succinctly explained below:

Algorithm 3 GS Markov Blanket

```
function GSMB($X, \mathbf{U}$)
    $S \leftarrow \emptyset$
    while $\exists Y \in \mathbf{U} - \{X\} : \; Y \not\perp X \mid \mathbf{S}$ do        ▷ Growing Phase
        $\mathbf{S} \leftarrow \mathbf{S} \cup Y$
    end while
    while $\exists Y \in \mathbf{S} : \; Y \perp X \mid \mathbf{S} - \{Y\}$ do        ▷ Shrinking Phase
        $\mathbf{S} \leftarrow \mathbf{S} - Y$
    end while
    return $\mathbf{MB}(X) \leftarrow \mathbf{S}$
end function
```

The algorithm is divided into two phases as described in the Algorithm 3: Growing Phase and Shrinking Phase. As the names suggest, in the first phase variables are added to the Markov Blanket of the variable X being evaluated and in the second phase variables are pruned.

In the first phase, Growing, for each one of the other variables in the domain (excluding X), if this variable Y is not independent of variable X given a set of variables $\mathbf{S}$, variable Y is added to the set $\mathbf{S}$. Notice that $\mathbf{S}$ is initially empty and represents the MB being constructed. Also notice, that tests for independence are made the way described before. In this phase, the algorithm tries to separate the variables, checking if they are conditionally independent. When conditional independence happens, they are not directly connected in the graph, since this conditional independence is represented in the structure by variables in the evidence between X and Y.

In the Shrinking phase the current MB is pruned by checking when variables X and $Y \in \mathbf{S}$ are conditionally independent given the other variables in the current blanket. Once again, the algorithm tries to separate variables, this time, using only variables from the output of the Growing phase, if there is conditional independence between X and Y the latter is removed from the MB. The result is a set of variables that belong to the MB of X. In the GS algorithm, after the GS Markov Blanket procedure has been executed for each variable, adjacent nodes are determined, cycles dissolved and edges oriented.

For more information about Grow and Shrink refer to [4].

2.4.3 Hybrid or Mixed algorithms

There are algorithms that mix characteristics of both approaches presented. One example of this is Max-Min Hill Climbing (MMHC) [7], that executes a score-based algorithm after reducing the search space using a constraint-based approach. In MMHC, Max-Min Parents and Children (MMPC) is used to determine the parents and children of the node in the Bayesian network. MMPC uses conditional independence tests to determine the parents and children set of the node in the Bayesian network graph. MMHC is described in [7].

After determining the PC set for each variable using MMPC, MMHC employs a score-based algorithm (Greedy Hill Climbing) with restrictions to only consider possible edges that connect two variables X and Y when Y is in the parents and children set of X. For these cases, as in GHC, the algorithm applies operations as adding, removing and inverting edges [7].

2.5 Local and Global Approaches

The Bayesian network structure learning algorithms may be divided into two groups: local and global learning. Local learning algorithms like SLL [19] and DMBC [10] where the search space does not include all possible Bayesian networks, restrict itself to only making changes to a smaller neighbourhood of a node, like the parents and children set of nodes, or the markov blanket of a given variable in the graph. Despite the recent local score-based approach presented in papers like [19], this approach, as cited in the same paper, is not new for constraint-based learning and had been presented in works like [26] and [27].

Other algorithms search the entire space of possible Bayesian networks. They consider all possible structures that could be formed by the set of all nodes in the graph, and use an heuristic to navigate through these networks. Consequently, they can make modifications to any edge in the graph. Global search algorithms are more common than the local approach ones, and are well represented by several known algorithms like K2, GHC, MMHC and GS. However, the global approach, because of its larger search space, is more computationally expensive, demanding more time to execute its algorithms.

2.6 Incomplete Data

A dataset is considered incomplete when there are missing values or hidden variables. A value is considered to be missing when in an example, the value for a given variable is not present. Also, a variable is considered to be hidden when for all examples in the dataset the value of the variable is not present. Missing values occur for a number of reasons, these include difficulties or failures in the process to gather the data [3]. The common approach adopted by several data mining workflows is not to consider these data by eliminating the examples where they occur or to use a technique where the dataset is completed. However, considering the pattern of missing data can lead us to a more unbiased solution, since data may not be missing at random, and a source or reason for these missing values may exist. Then, making an analysis of something in a completed dataset may lead to a biased conclusion. Example of reasons for missing data may include patient refusal to continue in a study, treatment failures or successes, adverse events, choice of not reporting critical statistics or fail to do so, any type of unavailability of data, and other common causes.

Unfortunately, a number of learning algorithms assume that data sets are complete. A possible cause is that inference in the presence of missing values is an expensive task, as shown in [3]. However, as cited before, research data usually have several incomplete examples. Thus, the hypothesis assumed by several algorithms gets a little bit unrealistic. Bayesian network structure learning from incomplete data imposes not only difficulties concerning what to do with incomplete examples, but also high computational costs when searching for an unbiased solution. As it can be seen in [3], assuming incomplete data imposes several foundational and computational problems for us to solve, like the increasing computational time as more missing values are present in the dataset.

An approach to deal with incomplete data in Bayesian networks is using the Expectation Maximization (EM) algorithm [3]. This algorithm requires the computation of the Expected Sufficient Statistics (ESS), the equivalent to the Sufficient Statistics (SS) when the data is complete. Computing sufficient statistics is fairly immediate since it usually requires only counting examples, but not with incomplete data when computing ESS. In this case, for each combination of possible values for variables that are missing in an example, inference is used. With the resulting probability values obtained from inference, each possible case is considered into the counting. This will be seen in more details when explaining EM. The point is that the number of necessary inferences fastly increases with the increase of missing data. And so the time to make all these inferences.

Another issue to consider is that when dealing with missing data, when using a score-based approach, since constraint-based ones does not allow incomplete datasets [28], several scores become not decomposable. When a score is decomposable, the total score of the network structure is a function of local scores involving the variable and its parents. Being decomposable is a desired characteristic for a score function, because in a score-based algorithm when comparing several network structures there is no need to recompute the score for every network being compared, but only for a smaller part where the change was made. Unfortunately, it does not happen if the score is not decomposable, and it is necessary to recompute all the score for each network being compared, and that is a problem when dealing with incomplete data.

2.6.1 Expectation Maximization

Expectation maximization (EM) [3] is a well known iterative algorithm that searches for the maximum likelihood estimates of the parameters of a Bayesian network. It is an algorithm used to learn parameters from a network and is able to deal with incomplete data. EM is composed by two steps: the first is called Expectation step and is responsible by computing Expected Sufficient Statistics that will be used in the second step called Maximization that uses the statistics obtained in the first step to estimate the parameters of the network. These two steps happen iteratively and it is guaranteed by the algorithm that the likelihood of the network always gets better at each iteration [3]. The Expectation step of EM is also useful in structure learning algorithms as the computation of the Expected Sufficient Statistics can be a form of trying to solve the problem of computing sufficient statistics [18], those are necessary in structure learning algorithms to compute the score.

Algorithm 4 Expectation Maximization

```
function EM($structure, data$)
    $Initialize\ parameters\ of\ structure$
    while $score\ improves$ do
        $ESS \leftarrow EM\_Expectation\_Step$
        $parameters \leftarrow EM\_Maximization\_Step(ESS)$
        $score\_calculation(structure)$
    end while
    return $structure\ with\ updated\ parameters$
end function
```

The expectation and maximization steps of the algorithm in Algorithm 4 is explained below.

First, before iterating through Expectation and Maximization steps, the parameters of the network are initialized. It is an important step since EM can get stuck on a local maximum if initialization is not good. And, considering that the likelihood function on the incomplete data case has more local maxima the more missing values exist, it becomes really important to initialize parameters correctly. There is no specific rule for this and initialization can be done by expert specification, by means of running another algorithm, or by doing several random initializations.

In the expectation step, the algorithm uses the current network with its current parameters to calculate the ESS, using the formula in Equation 2.9:

$$ESS(X_i, \mathbf{Parents}(X_i)) = \sum_{m=1}^{d} P(X_i, \mathbf{Parents}(X_i)|\mathbf{o}[\mathbf{m}]), \tag{2.9}$$

where, X_i is a variable of the Bayesian network, there are d examples, and $o[m]$ are the observed values in example m.

In the formula, calculated for each combination of values from a node and its parents, the probability of the assumed values for the nodes and its parents given the observed data on each example is computed. Then, the results for all the examples in the dataset is summed. Considering these calculations, notice that inference is made for all the examples in the dataset, for all the possible combinations of values from the variable and the values of its parents. As an illustration, suppose a dataset of 1000 examples, and a network of 3 variables A, B and C, where A and B are parents of C in the graph and there is no other edge. Consider also that all of them are discrete, binary variables. In this situation, the total number of parameters for ESS estimation is $2 \cdot 2 \cdot 2$ for C, plus 2 for A, plus 2 for B, times the number of examples in the dataset. At each iteration of EM, where these computations are needed, 12.000 inferences are made for this really small network, disconsidering any optimization.

When the examples are complete it is simpler to compute the ESS, because the result from the inference can only be 0 or 1. It is 0 if the query does not agree with the evidence and 1 if the opposite is truth. But when there is missing data, an algorithm for inference has to be executed, and the number of times it is executed makes the process time consuming.

After the Expectation step, it comes the Maximization one. At this step of EM, the parameters are estimated using the ESS computed before. It is done the same way as it is for sufficient statistics in the complete case. The counting for a given combination of values from the variable and its parents is divided by the total count for the parents values, resulting in Equation 2.10:

$$\hat{\theta}(X_i, \mathbf{Parents}(X_i)) = ESS(X_i, \mathbf{Parents}(X_i))/ESS(\mathbf{Parents}(X_i)). \tag{2.10}$$

After estimating each parameter, the likelihood score is calculated using the new parameters and the results are compared to the score using the old parameters. The difference between the two scores is computed and compared to a threshold given by the user. If the difference is smaller than the threshold given by the user, the algorithm stops. Otherwise, it continues to the next iteration.

One important and useful thing to notice about EM is that the likelihood of the network converges really fast in the first iterations, that is not possible to say the same about the parameters, but if the user is interested only in the likelihood increase, he can stop the algorithm with only a few iterations. After the first iterations, the algorithm continues to improve the likelihood but not so much as in the beginning, it is a slight improvement. However, the parameters can still suffer great variations.

2.6.2 Structural Expectation Maximization

Structural EM is an algorithm that allows Bayesian network learning from incomplete data [18]. It uses the Expectation step of the EM algorithm to reduce the problem of learning the structure of the network with incomplete data to the simpler problem of learning the network with complete data. When learning a Bayesian network with incomplete data, there are issues, as described before, concerning the decomposability of the score and the use of inference calculations to acquire the ESS. Since for each change made to the network the parameters need to be updated not only on the adjacencies of the node, requiring a new EM run, the procedure gets time consuming. SEM tries to deal with these problems by computing the ESS and using it as the sufficient statistics for a structural search step. It is an iterative algorithm and so after each structural search step, using an algorithm like GHC, parameters are reestimated using EM for the new structure and also the ESS.

Algorithm 5 SEM (Structural Expectation Maximization)

```
function SEM($structure, parameters, data, convergence\_threshold$)
    while $score\ improves$ do
        $ESS \leftarrow EM\_Expectation\_Step\ (structure,\ parameters,\ data)$
        $parameters \leftarrow EM\_Maximization\_Step\ (ESS,\ structure)$
        $structure \leftarrow Search\_Algorithm\ (structure,\ parameters,\ ESS,\ data)$
        $score\_calculation(structure)$
    end while
    return $structure,\ parameters$
end function
```

As shown in Algorithm 5, at each iteration the score is checked for improvement. If it does not improve enough the algorithm is stopped, but if so, it continues to the next iteration. Improvement criteria is determined by the user, using a threshold. At each iteration, EM Expectation step is run to compute ESS. The new parameters of the network that maximize the likelihood function are estimated during the Maximization step, using the same rules as for the complete case, but now using the ESS. These ESS are used in the score-based search algorithm to allow score calculation for each network. The scores are compared for all possible changes evaluated and the best network is chosen. This network will be the starting structure for the next iteration [18]. It is also possible to have more than one iteration of EM before entering the search for the structure in order to improve parameters even more, but likelihood rapidly increases in the first iterations [3], such that it is useful to consider only a few iterations.

The search algorithm used in SEM uses the ESS to estimate the parameters for each change and calculate the score. However, it is not recommended to keep reusing the same ESS after several changes in the structure [3], because they were estimated with the structure before entering the search algorithm and a new network would require a new parameter estimation procedure.

2.7 Scores

A score-based algorithm transforms the problem of learning a Bayesian network structure into the problem of searching the structure that scores the best, given a score function. This way it is very important to choose a good score function [3]. In this section the scores used at this work are described, but there are several other scores like BD [29], BIC [30], MDL [31], K2 [32], NML [33] and MIT [34].

2.7.1 Log-Likelihood

Maybe the most natural score, the likelihood score [3] is calculated computing the probability of the observed data given a set of parameters and a structure. The calculation is done by multiplying the probabilities of occurrence of each example, i.e., the probability of the variables in the domain to assume the values in the dataset example given the considered set of parameters and network structure. This may be expressed mathematically like this:

$$L(\mathbf{\Theta}|D) = P(D|\mathbf{\Theta}) = \prod_{i=1}^{N} P(\mathbf{X}_i|\mathbf{\Theta}). \tag{2.11}$$

In the formula, $\mathbf{\Theta}$ is the set of parameters θ of the Bayesian network, $\mathbf{X}_i$ is the set of values of the variables X for the example i, N is the number of examples and D is the dataset.

As the likelihood is a product of probabilities, when the dataset is sufficiently large, the decimal part of the number gets more and more algarisms, being difficult to represent in a computer. The solution is to compute the logarithm of the likelihood, since the log function assumes bigger values for bigger likelihood values, and the opposite is also true. As the interest is to compare likelihood values, this solution fits well to the problem. Also, it turns the product into a set of sums:

$$LL(\mathbf{\Theta}|D) = \log(P(D|\mathbf{\Theta})) = \sum_{i=1}^{N} \log(P(\mathbf{X}_i|\mathbf{\Theta})). \tag{2.12}$$

The likelihood function also has an information theory interpretation as a function of the mutual information and entropy [3]. When used as a score function in a score-based structure learning algorithm it presents some problems in the sense that it always gives a higher value for more connected networks. Then, if you have a structure S, and a structure S^{+1}, with one more edge, $L(S, \mathbf{\Theta}|D) \leq L(S^{+1}, \mathbf{\Theta}|D)$, with equality only being true in the rare case where there is perfect conditional independence between variables [3].

2.7.2 Conditional Log-Likelihood (CLL)

Conditional Log-Likelihood as presented in [35] computes the sum of the logarithms of the probability of a specific variable to assume the values in the dataset given all the other variables in the dataset. Unlike the likelihood, it concentrates in a specific variable, it is a score calculated for that variable. These probabilities are calculated given the parameters and structure of the Bayesian network, like in:

$$CLL(\mathbf{\Theta}|D) = \sum_{i=1}^{N} \log(P(\mathbf{X_i} = x_i|\mathbf{X} \setminus X_i; \mathbf{\Theta})). \tag{2.13}$$

Differently from accuracy, CLL not only gives us information about if examples were classified correctly or not. It also gives us information about how well the examples were classified. If CLL is used, from the formula, it is possible to notice that its not only a matter of doing classification correctly or not, but the probability of the classification being correct is also taken into account. Besides this, CLL also offers a way of evaluating the classification of only one variable, what is useful when trying to improve its classification.

2.7.3 Akaike Information Criterion (AIC)

The Akaike Information Criterion [36] is an information theory founded score (like BIC and MDL) that deals with the problem of evaluating how well a model fits to the data, as the other scores, but penalizing model complexity. This way, it surpasses problems like the one presented when computing the likelihood. Unlike the other scores, the smaller the AIC, the best the model is. AIC computation formula is shown below:

$$AIC = 2k - 2\log(L). \tag{2.14}$$

In the formula k is the number of parameters. The more connected a network is, the more parameters will be necessary to represent the network. L is the likelihood of the data to the structure.

Chapter 3

Bayes Ball Structure Learning

This section describes the algorithm that was developed in this work, called BBSL (Bayes Ball Structure Learning) [37]. BBSL is a local structure learning algorithm that is able to learn from incomplete data with focus on classification tasks. BBSL tries to improve the AIC_{CLL} score (to be presented in this chapter) for a specific variable given the data. It improves the classification of the variable without achieving an extremely complex model.

One of the advantages of working with a local approach is that it allows us to solve specific variable classification problems even in bigger networks. This is possible because this approach is usually faster than the global one, making changes only to a small set of edges. The global approach has a lot of possible Bayesian networks to consider during the search step and consequently takes more time. The local approach is also a recent approach in Bayesian network structure learning if compared to the older global approach, that has more published algorithms, as it will be presented in the Related Work chapter.

Also, incomplete data is a common characteristic of research datasets, and working with it is not a feature of many algorithms, even for the global case. Constraint-based algorithms usually are not able to support incomplete data, and until now, as far as we know, its unknown any constraint-based algorithm that can work with incomplete datasets. Moreover, for score-based learning, the usual approach is to complete the data before running an algorithm. This procedure usually inserts some more bias in the process. An alternative is to use the Structural Expectation Maximization (SEM) algorithm. It uses a global approach, and was adapted to build the algorithm in this work.

Making the union of the local approach with the SEM one for incomplete datasets takes the advantages of both. So that it allows us to create a new algorithm that is able to solve to some extent the problem of learning a Bayesian network structure locally, even if it is a big network, and even for datasets that are not complete. This is the problem that is being addressed here.

BBSL is a SEM-based algorithm, it makes some changes to the SEM Expectation Step, and proposes a local search algorithm to the Search Step. This local search uses only locally modified networks as candidates. These are generated based on the nodes returned by a previous step of BBSL. A big picture of BBSL is presented.

Algorithm 6 BBSL

```
function BBSL($data, class\_variable, structure, parameters, classif\_threshold,$
$count\_threshold$)
    while $score\_improves$ do
        $variables \leftarrow$ CollectRelevantVariables($\mathbf{data}, \mathbf{class\_variable}, \mathbf{structure},$
$\mathbf{parameters}, \mathbf{classif\_threshold}, \mathbf{count\_threshold}$)
        $ESS \leftarrow EM\_Expectation\_Step(structure, parameters, data, variables)$
        $parameters = EM\_Maximization\_Step(ESS, structure, variables)$
        $structure \leftarrow$ SelectStructure($\mathbf{structure}, \mathbf{parameters}, \mathbf{ESS}, \mathbf{data},$
$variables$)
        $score\_calculation(structure)$
    end while
    return $structure, parameters$
end function
```

The CollectRelevantVariables and SelectStructure steps of BBSL will be detailed later. By now, lets focus on the overall structure of the algorithm and changes made to the Expectation Step.

First of all, the structure is similar to the SEM structure presented before. It has an EM step and a structure search one. During the Expectation step statistics are computed from data, in the Maximization step parameters are estimated, and in the structure search every candidate network is evaluated in order to choose one that improves the score the most. During this search, networks scores are evaluated and compared to determine if the algorithm should continue to the next iteration or not. Notice that the minimal improvement threshold to get to the next iteration is a parameter of SEM and needs to be determined previously.

In order to select a restricted set of variables that will be used in BBSL, such that it can work locally, there is the CollectRelevantVariables step. From the output of CollectRelevantVariables, it comes a set of variables that is able to affect the probability distribution of one specific variable. This variable, also a parameter of the algorithm, is the variable whose classification needs to be improved. This specific variable will be called the class variable, since only classification problems are being considered.

The CollectRelevantVariables set is the group of variables that is able to change the probability distribution of the class variable considering if variables are observed or not in the examples and if the class variable is missclassified or not for each example. The algorithm to obtain it will be explained and detailed in Section 3.1.

This set of variables, output of the CollectRelevantVariables step, is used to make a local version of Expectation Maximization algorithm. Expectation Maximization usually compute Expected Sufficient Statistics (ESS) for all the variables in the Bayesian network. This demands too much computational resources for big networks and high number of examples. The local version of EM used here does not include all the variables in the Expected Sufficient Statistics computation. Using this local EM approach affects the behaviour of the Expectation and Maximization steps by reducing the number of ESS to be computed and parameters to be estimated.

Considering this, the performance will be dependent of the number of variables only in the likelihood score calculation, used at each iteration of the EM, since the other steps are local. The ESS needed to the next steps will be calculated only for the variables in the surroundings of the class variable that affect the most its classification, represented by the output of the CollectRelevantVariables step. As the ESS calculation demands too much computational resources to be computed, calculating it to only a few variables makes the algorithm to be executed faster.

The several ESS calculations for each variable are independent of its calculations for other variables, so this computation can be parallelized. It is also possible to divide the dataset, compute the ESS for parts of the dataset and sum the results in the end. Taking advantage of this characteristic is not something new in the literature, as shown in [38], and it was done here since the most modern computers have several cores, and it is possible to take advantage of this feature using threads to allow faster computation.

ESS are computed for each combination of values of a variable and its parents. When searching for a structure the score-based way, it would be necessary to recalculate the ESS for each candidate change in the network, because a modification in one edge of a network changes the parents of one or two variables. Recomputing ESS for all the variables, in this case, would not be good to the overall execution time. When you make a change to the structure of a Bayesian network, the specific set of ESS that were calculated to the old Bayesian network structure does not apply to the new structure, because ESS is calculated for each set of variables and its parents, or in other words, for each line of each CPT. In order to avoid recalculating all ESS, only the ESS for the affected CPTs are recalculated.

Variables whose parents are changed during search, in BBSL, are part of the CollectRelevantVariables set (usually a small set of variables). Changes in the structure are made only between variables of this set, as it will be explained in Section 3.2. Considering this, the ESS for the variables in the CollectRelevantVariables set and its parents is calculated before entering the search, as a result of the local expectation step. Then, it is easier to compute the ESS for a network that suffered a local change, by computing the ESS to the variables affected by the structural change. All the ESS for the other variables in the CollectRelevantVariables set is already available during the structure search.

To estimate the parameters from the network, the maximization is done once or iteratively with the expectation step, depending on the implementation, both are possible. Iterations are done in order to improve the likelihood score of the Bayesian structure network and learn the parameters. Parameters are also learned because it is interesting to have in the end of all the procedure a set of structure and parameters that allows someone to make inference about the variables in the domain. Notice that, not all structure learning algorithms return a network that allows one to make inference, because, for instance, there are no parameters, resulting from only learning structure, as in GS and PC.

At this maximization step it is also used a local approach so that parameters that were not changed are repeated from the previous structure, and the changed parameters, inside the CollectRelevantVariables set, are estimated using the same technique used for complete data. For the computation of the parameters, the ESS calculated before, in the Expectation step, is used.

After the maximization, the search for the structure is made locally through SelectStructure which returns the best structure accordingly to the AIC_{CLL} score that will be explained later, using the heuristic presented in the algorithm.

In the end, the score is calculated to define if the algorithm continues or not to the next iteration of BBSL. The score calculation step in the algorithm is just an emphasis resource to clarify what is being done, since the score could be stored from the SelectStructure search step. After the loop, BBSL returns a Bayesian network, with its structure and parameters, and desirably this structure will be able to classify the class variable better than before.

3.1 The First Step of BBSL (CollectRelevant-Variables)

The first step of BBSL, referred as CollectRelevantVariables before, receives as input a Bayesian network and a class variable. Then, it returns the set of variables that can affect the class variable distribution for part of the missclassified examples. The variables returned are not conditionally independent of the class variable given the observed variables for several examples in the dataset. Algorithm 7 shows how it works.

In the algorithm, first, the set of relevant variables to the classification task is initialized as empty. This is the set that will be filled and returned in the end of the algorithm for this step.

For each example in the dataset, the value for the class variable in the example is checked to determine if it is observed or not, if the variable is not observed, this example will not be considered in the evaluation. However, if the class variable assumes any value at this example, then the algorithm calculates the probability of the class variable to assume the value given the other observed values in the same

Algorithm 7 CollectRelevantVariables step

```
function CollectRelevantVariables($Dataset$, $ClassVariable$, $BN$,
$classif\_threshold$, $count\_threshold$)
    $RelevantVariables \leftarrow \emptyset$
    for all $example$ in $Dataset$ do
        if $ClassVariable$ is observed then
            $Prob = P(ClassVariable = c \mid \textbf{observed}[example] = \textbf{o})$
            if $Prob \leq classif\_threshold$ then
                $BB \leftarrow BayesBall(ClassVariable, example, BN)$
                $RelevantVariables\ \cup BB$
                $Increment\ count[node]\ for\ each\ node\ in\ BB$
            end if
        end if
    end for
                                                    ▷ Filter nodes in RelevantVariables
    for all $node$ in $RelevantVariables$ do
        if $count[node] \leq count\_threshold$ then
            $RelevantVariables.remove(node)$
        end if
    end for
    return $RelevantVariables$
end function
```

example. Here, the algorithm is trying to evaluate how well the classification task is being done for this specific example. This probability calculation is done the same way it is a single term of the sum in CLL.

If the result from the previous calculation is good (it is greater than the classification threshold), so that the classification is being well done, the algorithm considers that there is no need to improve it. Classifying the result in good or bad depends on a classification threshold that is a parameter of the algorithm and needs to be determined before its execution. If the classification is bad, and that means the same as if the result is below the threshold for BBSL, then it is necessary to improve it.

If the result of the classification is bad, BBSL tries to determine the set of variables that are relevant to the classification of the class variable using Bayes Ball. Now, that it is already known that the example is not classified as well as the user would like, BBSL tries to detect which are the variables that may have caused this misclassification using Bayes Ball. This is done for each example in the dataset that is misclassified, since this algorithm is oriented by the classification results found for the examples.

The solution proposed, to run the Bayes Ball algorithm presented before, allows to determine, given the states of observation of the variables and a class variable, which are the variables that may influence its distribution. By states of observation, it is meant if the variable is observed or not. It means, if the variable assumes a value in that specific example, such that it is placed in the evidence set for Bayes Ball. This way, Bayes Ball is able to return a set of variables that may influence the classification of the variable. Notice that, although the Bayes Ball algorithm is used here, any other algorithm with the same objectives could have been used, any algorithm that would return the set of variables that are not d-separated from the class variable.

The procedure described before is done for each example in the dataset and its result is united to the previous set obtained in the loop, forming a bigger set with the union of all sets of variables returned by the Bayes Ball on each example evaluated.

As BBSL always makes the union of the sets returned by Bayes Ball in the loop, it may happen that one example, given the state of observation of its variables, when submitted to Bayes Ball, returns a big number of variables. This number may be considered big relatively to the size of the network or to the Bayes Ball return for other examples. In order to avoid that a small set of examples be responsible by the inclusion of a big set of variables, degrading performance, filtering is necessary. If the number of variables is big, more combinations of variables are possible, and consequently, more structural modifications, and more candidate networks are considered in the search. As estimating parameters and computing a score is needed for each candidate network, and these tasks demand more computational resources, performance is degraded.

In order to filter these variables, for each variable in the Bayesian network is associated a counter. Each counter registers the number of times a variable was returned by the Bayes Ball algorithm. It works like if the examples voted for which variables should enter the returned set, since at each misclassified example, Bayes Ball is run and a new set returned. This number associated to each variable will be considered later in a filter.

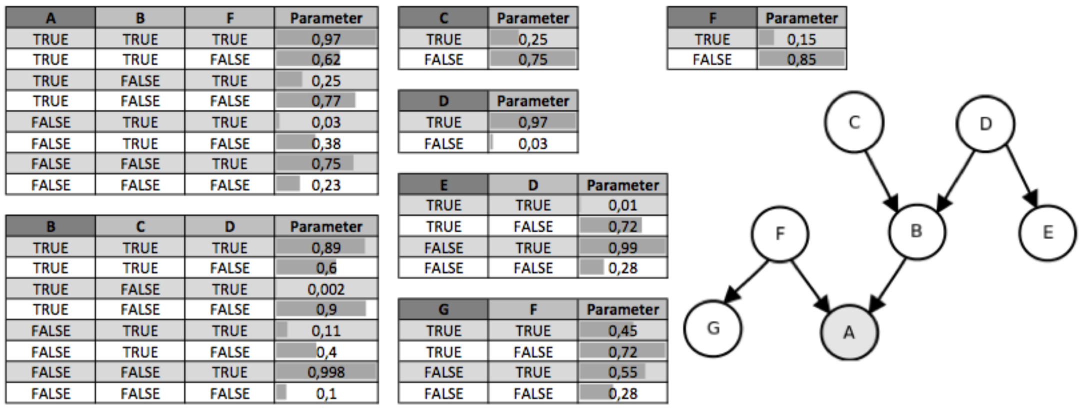
When filtering, for all nodes in the set of relevant variables to the class one, the algorithm checks if the counter associated with the variable registers a value that is greater than a threshold. This value is also a parameter to the algorithm, and as any other parameter in this algorithm it should be determined before its execution.

The value of the threshold is a percentual, the minimum quantity of times a variable should be selected by the Bayes Ball algorithm, divided by the total examples considered in the evaluation. Variables which the counter is below the threshold are eliminated from the relevant variables set.

The result of this filtering is the returned set of variables of the first step of the algorithm, and that is what allows the local approach considered here. This set is used when computing the ESS and in the search procedure. It allows improvements in the computation time, and a better focus on the classification problem. This focus is obtained by not considering variables that would not have a strong impact to the classification results.

3.1.1 Procedure Illustration

In this section, it is presented an example so as to illustrate the procedure explained above. For this reason an artificial network was built and is presented in Figure 3.1. In the picture, the structure of the Bayesian network is presented together with the Conditional Probability Distribution of the variable. Read the table as the probability of the first variable in the table from left to right, given the probability of the other variables in the table. The lines of the table show the values assumed for each variable and the parameters are the conditional probabilities values.



A	B	F	Parameter
TRUE	TRUE	TRUE	0,97
TRUE	TRUE	FALSE	0,62
TRUE	FALSE	TRUE	0,25
TRUE	FALSE	FALSE	0,77
FALSE	TRUE	TRUE	0,03
FALSE	TRUE	FALSE	0,38
FALSE	FALSE	TRUE	0,75
FALSE	FALSE	FALSE	0,23

C	Parameter
TRUE	0,25
FALSE	0,75

F	Parameter
TRUE	0,15
FALSE	0,85

D	Parameter
TRUE	0,97
FALSE	0,03

E	D	Parameter
TRUE	TRUE	0,01
TRUE	FALSE	0,72
FALSE	TRUE	0,99
FALSE	FALSE	0,28

B	C	D	Parameter
TRUE	TRUE	TRUE	0,89
TRUE	TRUE	FALSE	0,6
TRUE	FALSE	TRUE	0,002
TRUE	FALSE	FALSE	0,9
FALSE	TRUE	TRUE	0,11
FALSE	TRUE	FALSE	0,4
FALSE	FALSE	TRUE	0,998
FALSE	FALSE	FALSE	0,1

G	F	Parameter
TRUE	TRUE	0,45
TRUE	FALSE	0,72
FALSE	TRUE	0,55
FALSE	FALSE	0,28

Figure 3.1: Bayesian Network example

Also consider this small artificial dataset with only 10 example for this illustration. In the table it is also presented the probability of the class variable to assume the observed value in the example, given that the other variables assumed the observed values in the same example.

Table 3.1: Example of dataset

	A	**B**	**C**	**D**	**E**	**F**	**G**	*Probability*	*BayesBall*
1)	True	True	False	True	?	True	True	0,97	A, B, F
2)	False	?	False	False	True	False	?	0,37	A,B,C,D,F
3)	True	True	?	False	True	?	True	0,65	A,B,C,D,F
4)	True	True	False	?	False	True	?	0,97	A,B,F
5)	True	?	?	?	?	?	?	0,69	A,B,C,D,E,G,F
6)	False	True	False	False	?	False	False	0,38	A,B,F
7)	True	?	False	True	False	False	True	0,77	A,B,C,D,F
8)		True	?	False	True	False	False	?	A,B,F
9)	True	True	False	?	False	True	True	0,97	A,B,F
10)	False	True	False	True	?	True	False	0,03	A,B,F

Following the algorithm, for each example in the dataset, it is applied a sequence of steps. For the example, consider the class variable to be A, and the classification threshold to be 0.7.

Lets consider the first example in the dataset, the probability of the class variable A to assume value $True$ given that B, C, D, F and G assumed values $True$, $False$, $True$, $True$ and $True$ respectively is computed. The result is 0.97, as shown in the table. This value is compared to the classification threshold, 0.7, and, as its greater than it, it is considered that there is no need to improve this classification and the algorithm follows to the next example.

For the second example, inference is made considering the new values for the variables, and the result is 0.37. The calculations, as in the first case, are made considering the current initial structure and parameters of the Bayesian network. Now, that the inference result is lesser than 0.7, it is necessary to improve classification.

First of all, the nodes that affect classification are computed using the Bayes Ball algorithm (they are the nodes that can affect A probability distribution). For this example, as shown in Table 3.1, the variables returned from Bayes Ball execution are A,B,C,D and F. The counter from each of these variables is incremented by 1, and the total of examples is also updated.

The same is done for the third example, notice that depending on the state of observability of the variables, using the concept of d-separation, Bayes Ball returns a different set of variables. The fourth, fifth, sixth and seventh follow the same rules, incrementing the counter for the variables returned by the Bayes Ball when the value returned by inference is greater than the previously established classification threshold.

However in the eighth example, something different happens, the class variable A has no value. When it happens, the algorithm simply does not consider the example.

By the end of the execution the following counts were computed for this case:

Table 3.2: BBSL Counts

A	**B**	**C**	**D**	**E**	**F**	**G**
5	5	3	3	1	5	1
100%	100%	60%	60%	20%	100%	20%

Now consider the filtering step that happens after the loop. If a limit of 60% to the acceptance of the node to the next step of BBSL is imposed (user parameter), then, for this special case, only variables A, B and F would be considered. It is a high value, and almost all other variables would enter the returning set if the value was reduced to 20%, except E and G. Although this has happened in this example, in a larger network, where a greater number of nodes are considered, some variables would never be counted and some would have a small percentual caused by special and rare cases like the one where all the variables in the Bayesian network are not observed, except the class variable (like in the fifth example of the dataset).

In the end of this step of the algorithm, this filtering was done to avoid that if only one example like this occurs in thousands of examples in a dataset, not a lot of variables need to be included. It avoids that several variables be included only to enhance the classification performance for a small portion of the dataset. This makes the algorithm to be attained to the bigger classification improvements.

3.2 The Second Step of BBSL (SelectStructure)

After the EM step, in the SEM algorithm, there is a structure search step where the results from previous steps are used. Results from previous steps include the

ESS calculated from the current structure and parameters, using the dataset, and the current Bayesian network. In BBSL something similar is done, but a specific behaviour is proposed.

In BBSL, the entries for this second step are composed by: a) the set of variables determined in the first step of the BBSL, using the Bayes Ball algorithm; b) the class variable; c) the set of ESS originated from the EM step, only making changes to the ESS selected by the first step of BBSL; d) the current structure of the network; e) the estimated parameters from the maximization step; and the f) dataset considered.

Algorithm 8 SelectStructure step

```
function SelectStructure($RelevantVariables, ClassVariable, Data, BN$)
    $Neighbours \leftarrow GenerateNetNeighbours(RelevantVariables, BN)$
    $BestBN \leftarrow Evaluate\ AIC_{CLL}(Neighbours, Data, ClassVariable)$
    return $BestBN$
end function
```

From these entries, the BBSL second step is very similar to one iteration of Greedy Hill Climbing, except by the fact that not all possible structures are evaluated. Only a small set of nodes is considered in the algorithm to form a new structure. With this restriction, the algorithm execution takes less time and allows it to be run for bigger Bayesian networks.

The algorithm is described in Algorithm 8. First, from the current structure, the candidate networks are generated. The candidate networks are neighbour networks that only differ from the current structure from a local structural change.

A neighbour network is a Bayesian structure that differs from the current structure only by the application of one operator. Here, the operators considered are the same as in GHC: a) add an edge; b) remove an edge; or c) invert an edge. The restriction applied to the neighbours is that the change needs to be local. The application of only one operator, between variables that are in the set returned by the first step of the BBSL algorithm (CollectRelevantVariables), is considered. Therefore, it is only allowed to add, invert or remove edges between variables that are in this set of variables returned by the first step of BBSL.

After generating the candidate networks, each of them is evaluated using the AIC_{CLL} score, presented later. In the evaluation it is necessary to compute the CLL for the network. For this, after changing the structure, new parameters for the network are estimated using the ESS calculated before entering the search. New parameters estimation and AIC_{CLL} computation is done for each network. After each score computation, the structure that minimizes it, is selected to enter the next iteration.

It is important to notice that only one change is applied to the network, applying more changes at each iteration is also possible, however it is not recommended to change the network very much, since ESS are the same from the beginning of the search procedure and needs to be updated [3].

3.3 AIC_{CLL} score

When using the CLL score in the experiments, it was noticed that networks chosen by BBSL and GHC were very connected. In [3], some problems of using the likelihood score to select Bayesian networks are presented. In the same work, it is demonstrated that networks selected using the likelihood score tends to run into problems when working with new instances, a desired characteristic of machine learning algorithms. As demonstrated there, there is some preference for more connected networks, causing overfitting.

The log-likelihood score may be decomposed into CLL and one more sum of terms, as presented in [35]:

$$LL = \sum_{d=1}^{n} \log P(Y_d, \mathbf{X_d}). \tag{3.1}$$

In Equation 3.1, the log-likelihood formula is presented in a more adequate way to perform the decomposition using the CLL. In the formula, n is the number of examples in the dataset, d is one particular example, Y_d is the class variable and $\mathbf{X}$ is the set of other variables in the Bayesian network. The decomposition is performed like this:

$$LL = \sum_{d=1}^{n} \log(P(Y_d|\mathbf{X_d}) \cdot P(\mathbf{X_d})); \tag{3.2}$$

$$LL = \sum_{d=1}^{n} \log P(Y_d|\mathbf{X_d}) + \log P(\mathbf{X_d}); \tag{3.3}$$

$$LL = \sum_{d=1}^{n} \log P(Y_d|\mathbf{X_d}) + \sum_{d=1}^{n} \log P(\mathbf{X_d}); \tag{3.4}$$

$$LL = CLL + \sum_{d=1}^{n} \log P(\mathbf{X_d}). \tag{3.5}$$

In [35], it is explained that the second term when calculating the log-likelihood usually dominates the calculation.

In order to justify our next steps, our choice for the AIC_{CLL} score proposed here, it is shown, in a demonstration someway analogous to the one presented by [3], that CLL score also may prefer more connected networks. As this was not found in the literature, it will be presented here.

For this, imagine a graph G_1, where variable Y is not conditionally independent of the set of variables $\mathbf{W}$ given the set of variables $\mathbf{X}$. Also, imagine a graph G_0 where this conditional independence holds. The difference between the CLL scores of both graphs is:

$$CLL_{G_1} - CLL_{G_0} = \sum_{d=1}^{n} \log P_1(Y_d|\mathbf{X_d}, \mathbf{W_d}) - \sum_{d=1}^{n} \log P_0(Y_d|\mathbf{X_d}, \mathbf{W_d}). \tag{3.6}$$

If $M[y, \mathbf{x}, \mathbf{w}]$ are the counts for how many times $Y, \mathbf{X}$ and $\mathbf{W}$, assume $x, \mathbf{y}, \mathbf{w}$ values respectively. Then, the sum can be changed to:

$$CLL_{G_1} - CLL_{G_0} = \sum_{y,\mathbf{x},\mathbf{w}} M[y, \mathbf{x}, \mathbf{w}] \cdot \log P_1(y|\mathbf{x}, \mathbf{w}) - \sum_{y,\mathbf{x},\mathbf{w}} M[y, \mathbf{x}, \mathbf{w}] \cdot \log P_0(y|\mathbf{x}, \mathbf{w}). \tag{3.7}$$

And noticing that $M[y, \mathbf{x}, \mathbf{w}] = n \cdot \hat{P}(y, \mathbf{x}, \mathbf{w})$ then:

$$CLL_{G_1} - CLL_{G_0} = \sum_{y,\mathbf{x},\mathbf{w}} n \cdot \hat{P}(y, \mathbf{x}, \mathbf{w}) \cdot log P_1(y|\mathbf{x}, \mathbf{w}) - \sum_{y,\mathbf{x},\mathbf{w}} n \cdot \hat{P}(y, \mathbf{x}, \mathbf{w}) \cdot log P_0(y|\mathbf{x}, \mathbf{w}). \tag{3.8}$$

In the case where conditional independence holds, then:

$$CLL_{G_0} = n \cdot \sum_{y,\mathbf{x},\mathbf{w}} \hat{P}(y,\mathbf{x},\mathbf{w}) \cdot \log P_0(y|\mathbf{x},\mathbf{w}) = n \cdot \sum_{y,\mathbf{x},\mathbf{w}} \hat{P}(y,\mathbf{x},\mathbf{w}) \cdot \log \hat{P}(y|\mathbf{x}). \quad (3.9)$$

So, making this substitution, and joining the sums:

$$CLL_{G_1} - CLL_{G_0} = n \cdot \left(\sum_{y,\mathbf{x},\mathbf{w}} \hat{P}(y,\mathbf{x},\mathbf{w}) \cdot (\log \hat{P}(y|\mathbf{x},\mathbf{w}) - \log \hat{P}(y|\mathbf{x})) \right). \quad (3.10)$$

Applying the logarithm properties:

$$CLL_{G_1} - CLL_{G_0} = n \cdot \left(\sum_{y,\mathbf{x},\mathbf{w}} \hat{P}(y,\mathbf{x},\mathbf{w}) \cdot \log \left(\frac{\hat{P}(y|\mathbf{x},\mathbf{w})}{\hat{P}(y|\mathbf{x})} \right) \right), \quad (3.11)$$

and rewriting:

$$CLL_{G_1} - CLL_{G_0} = n \cdot \left(\sum_{y,\mathbf{x},\mathbf{w}} \hat{P}(y,\mathbf{x},\mathbf{w}) \cdot \log \left(\frac{\hat{P}(y,\mathbf{x},\mathbf{w})}{\hat{P}(y|\mathbf{x}) \cdot \hat{P}(\mathbf{x},\mathbf{w})} \right) \right); \quad (3.12)$$

$$CLL_{G_1} - CLL_{G_0} = n \cdot \left(\sum_{y,\mathbf{x},\mathbf{w}} \hat{P}(y,\mathbf{x},\mathbf{w}) \cdot \log \left(\frac{\hat{P}(y,\mathbf{x},\mathbf{w})}{\hat{P}(y|\mathbf{x}) \cdot \hat{P}(\mathbf{w}|\mathbf{x}) \cdot \hat{P}(\mathbf{x})} \right) \right); \quad (3.13)$$

$$CLL_{G_1} - CLL_{G_0} = n \cdot \left(\sum_{y,\mathbf{x},\mathbf{w}} \hat{P}(y,\mathbf{x},\mathbf{w}) \cdot \log \left(\frac{\hat{P}(y,\mathbf{w}|\mathbf{x})}{\hat{P}(y|\mathbf{x}) \cdot \hat{P}(\mathbf{w}|\mathbf{x})} \right) \right). \quad (3.14)$$

The factor that multiplies n is the Conditional Mutual Information ($I(Y,W|X)$), making this last substitution:

$$CLL_{G_1} - CLL_{G_0} = n \cdot I(Y,W|X). \quad (3.15)$$

Conditional Mutual Information is always non-negative, so or $CLL_{G_1} = CLL_{G_0}$, or $CLL_{G_1} \geq CLL_{G_0}$. The conclusion is that CLL score prefers networks where conditional independence are not present for some group of nodes, being equal only in the case where Conditional Mutual Information is perfectly zero, that means, when nodes are totally conditionally independent, what is unlikely to happen due to statistical noise.

When the $\mathbf{X}$ set is empty, the formula above reduces itself to the mutual information between $\mathbf{X}$ and Y. And again, the mutual information is also nonnegative, and it has as a consequence that the demonstration above applies to the independence case between variables or set of variables, besides the conditional independence case.

In order to avoid the problem of overfitting due to high connectivity of the network considering the motivation above, it would be interesting to have some kind of penalization for complexity. This characteristic is present in the AIC (Akaike Information Criterion) score. Considering that it could be possible to use it with CLL, it was preferable to do so. This score, to which is referred as AIC_{CLL} is proposed in Equation 3.16:

$$AIC = 2k - 2CLL. \tag{3.16}$$

Notice that the only change made to the AIC score, was the substitution of the log-likelihood (LL) score in the parentheses for the conditional log-likelihood (CLL). This new score has the local characteristics of the CLL, and the penalization for complexity in function of the number of the parameters of the AIC score.

The same way as the AIC score, the AIC_{CLL} score gets better the lesser is the score value. It means, by this criterion, that a better model is chosen if the AIC_{CLL} score is lower.

3.4 Other Approaches

When trying to improve BBSL, some changes in it were tested, some of then are presented in this section.

3.4.1 External Edges

Analyzing the proposed algorithm, BBSL, it is possible to notice a case for which the algorithm does not apply. It occurs in the specific case the class variable is independent of all others in the graph (when the class variable is not connected to the others). The Bayes Ball algorithm returns the set of variables that may influence the probability distribution of the class variable given the states of observability of the variables in the Bayesian network. However, if the class variable is independent of all other variables there is no way it could return a single node more than the class variable itself. This way, the first step of BBSL only returns the class variable and the search step does not produce any candidate networks. As a consequence, the network is not changed and the algorithm stops without making any modification.

To solve the problem cited above, it was proposed the addition of one more operator to the search step, the "add external edge" operator. This allows the algorithm to consider, besides the other three operators, this fourth one that allows edges from the class variable, to any other nodes in the network. That is, using this operator, it is allowed during the search step, that one edge from any node to the class variable, or, from the class variable to any node, to be added to the graph.

This has the drawback of adding more candidate networks to the evaluation step, thus increasing computation time. But it solves the problem of connecting an independent class variable to the other nodes of the graph, allowing bayes ball to be more useful in this specific case. In the next iterations, it would be possible to the class variable to find better connections.

3.4.2 Lidstone Smoothing

While trying to solve the problem of the overfitting caused by running the algorithm using only the CLL, i.e., without using the AIC_{CLL} score, one option to improve the performance of the classifier on new instances was to cause a perturbation to the parameters of the network. This was done since it was observed that for the test set, some classifications were done very well, while others were poor. The impact of the small probabilities in the logarithm, consequence of bad classifications in the CLL score, used to decrease the score very much, even for a small number of instances of the test set. The solution, before thinking in AIC_{CLL}, was to cause a small perturbation to the parameters of the Bayesian network returned by BBSL. This way, the effect of overfitting used to be atenuated. The technique used to cause this perturbation, keeping the parameters consistent to the probability rules was the Lidstone Smoothing [39]. The formula is shown in Equation 3.17:

$$\theta_{new} = \frac{\theta_{old} + \alpha}{1 + N * \alpha}. \tag{3.17}$$

In the formula, α is an additive parameter for smoothing and N is the number of possible values to be assumed by the variable.

Chapter 4

Experimental Results

In this chapter, the methodology applied to make the experiments is presented, as well as the results obtained from the experiments execution. Results are discussed on Section 4.4 and its specific conclusions are shown. The results contain statistics from CLL, Time spent, AIC_{CLL} optimization score and SHD. More general conclusions are left to the Conclusion chapter. Some more detailed tables with results are presented in the Appendix of this work.

4.1 Algorithms Used in Comparisons (DAHVI, SEM_{GHC}, GS and MMHC)

Before discusssing the results obtained, it would be usefull to justify some choices made considering the algorithms used in the comparison. As presented in the Related Work chapter, there are several possibilities of algorithms, however not all of them fit the requirements needed for a good comparison of BBSL with existing techniques.

First, to achieve a fairer comparison, it would be desirable to compare BBSL with other algorithms that have the same proposal as it has. The best scenario would lead us to compare BBSL with other local learning, for incomplete data algorithms, that focused on classification tasks. Unfortunately, as far as it goes our structure learning knowledge, the only algorithm that fits this description is DAHVI. DAHVI is a revision algorithm that tries to improve the classification of a group of variables by hidden variable insertion.

An alternative to try to solve the problem of selecting algorithms for comparisons could be to use local learning algorithms, as BBSL. However, there is the need to ignore other restrictions like focus on classification tasks or working with incomplete data, because there are only a few algorithms available that are local and also have these characteristics. As far as we know, options include SLL and DMBC. Part of these options also were not published at the beginning of this research, what brings some difficulties. However, even these algorithms do not allow using incomplete data. SLL does not allow it because of its OptimalNetwork heuristic presented in [40], that needs the score to be decomposable. And DMBC, as a K2-based algorithm, has some limitations also present in K2 , in what concerns its score.

Not considering the local approach restriction, there are algorithms that could be used for incomplete data, these include SEM and its variations. However, SEM is not local as BBSL and not even focus on classification tasks.

Because of this, for this research, it was decided to use a constraint-based algorithm, a score-based algorithm and a hybrid algorithm for comparison. A representative member of each group of algorithms has been chosen.

4.1.1 Constraint-Based Choice

For the constraint-based cited in the Related Work chapter, GS and PC are algorithms that return a global structure for the Bayesian network. This is better for our purposes because a full network is also the output of BBSL and allows fairer score comparisons. GS and PC do not return only a local set of variables as many constraint-based solutions. GS was selected for the comparison.

4.1.2 Score-Based Choice

From the score-based algorithms, the choice was SEM, because of being able to work with incomplete data. It is a common approach to use EM or some kind of adaptation of it to learn in an incomplete data scenario. However, it is necessary to determine a search algorithm to use in the SEM framework. For simplicity, the one selected was GHC, and this solution is called here SEM_{GHC}. At this work, it was used the SEM algorithm with the GHC in the search step.

4.1.3 Hybrid Choice

MMHC was chosen to represent the hybrid algorithms because of being recent, if compared to other structure learning algorithms like H2PC. And also, because of being popular, since it has been cited in several works (at the time of writing this dissertation H2PC was cited only once, while MMHC, 420, considering data from Google Scholar).

4.2 Required Algorithm Adaptations

From the selected algorithms DAHVI, GS, MMHC and SEM_{GHC}. In order to perform comparisons, GS, MMHC and SEM_{GHC} had to be adapted.

For SEM_{GHC}, the bigger search space of a global approach united to the incomplete dataset computational cost problems made SEM with GHC not to run in a time slice it could be possible to wait. This execution time depends on infra-structure restrictions that could not be surpassed. Also, this time consumption increases with the size of the dataset, number of variables and percentual of examples containing missing data. The alternative to these performance problems was to change the search algorithm a bit in pursuance of a better execution time. While GHC normally compares all the possible Bayesian network neighbours, the algorithm used in the comparison has the same heuristic as GHC, but compares only 50 randomly chosen neighbours of the current structure.

For the constraint-based algorithms there is an issue. One of the disadvantages of using a constraint-based algorithm is that they do not usually allow incomplete data [28]. None of the constraint-based algorithms cited in the Related Work chapter can deal with incomplete data and the interested reader can check this at each algorithms respective article.

The workaround found to deal with the problem of using incomplete data with constraint-based algorithms, trying not to change the structure of the algorithm, was to compute the ESS for the dataset before any change to the network. Then, before executing GS, that is constraint-based, the ESS were computed for the dataset and the available initial network. After ESS computation, the algorithm was executed using these ESS when statistics were needed. In the end, parameters of the learned

Bayesian network are estimated from these ESS. This is not the recommended scenario, since changes will be made to the network and the ESS are not updated [3]. But, it is an option that allows computation in a reasonable time.

MMHC, the hybrid option chosen, is also not able to work with incomplete data because of its constraint-based initial part. The same adaptations made to GS (constraint-based) were made to MMHC.

Remember that MMHC and GS start from a totally disconnected network. BBSL does not have this restriction, but Bayes Ball is not useful when all the nodes are independent of each other. Anyway, algorithms were compared. GS and MMHC starting from disconnected networks, and BBSL starting from a generated one.

Probably, one of the hardest parts of preparing the experiments involved choosing the algorithms (if infrastructure issues are disconsidered), because of these differences in the characteristics between them and BBSL.

4.3 Experimental Configurations

For all the experiments done in this work the computers had processors Intel i7, 8GB of RAM memory, and had Ubuntu as operational system. This is exposed because it affects the performance of implemented algorithms concerning the time they take to be executed, and allows more reproducibility. All the code used was implemented in Python 2.7.3, and all the algorithms were implemented without using any Bayesian network framework for this. All the algorithms implementations followed the instructions of the articles where they were published. The inference algorithm used in this work was Variable Elimination [3].

As justified before, the algorithms chosen for comparison were the constraint-based algorithm Grow and Shrink, the hybrid Max-Min Hill Climbing and the score-based GHC inside an SEM structure (with some modifications). They are commonly used in the literature for comparisons, have its pseudo-code available and implementable, and were considered representatives of each category to which each of them pertain to.

Four different networks were used in this work, all of them are commonly used in the literature, in special the ALARM network, present in several papers. The

ALARM (A Logical Alarm Reduction Mechanism) is a real life Bayesian Network, described as a network to monitor patients in intensive care [41]. Some of these networks may be found in http://www.bnlearn.com/bnrepository/. These networks include: ALARM (37 vertices, 46 edges, 509 parameters), Child (20 vertices, 25 edges, 230 parameters), Hailfinder (56 vertices, 66 edges, 2656 parameters) and Insurance (27 vertices, 52 edges, 984 parameters).

For each network, artificial datasets were generated by means of using the Forward Sampling [3] method. In forward sampling, each variable in the network is sampled in topological order, until there are no more variables to be sampled. The result of this sampling is an example in the dataset. The procedure is repeated until filling the quantity of examples needed for the artificial dataset. The choice for building artificial datasets from these known networks was done because of the difficulties to find real world datasets that filled the needs of the research, and for the lack of time for collecting data to build real world datasets.

From each of these artificial datasets generated, that had 500 and 1000 examples, other four datasets were generated changing the amount of missing data in it. Finally there were a set of 8 datasets, 4 for 500 examples and 4 for 1000 examples. Each group of four, contained datasets of 1%, 5%, 15% and 30% of examples that had at least one missing value.

The networks cited above were also used to generate new networks. They were 5 different networks, each of them with three random local changes to its structure, resulting in a total of 20 networks to be tested. These local changes were applied adding, removing or inverting edges that were 2 hops at maximum from the class variable. Networks were generated this way in order to try to simulate an hypothetical expert that would elicitate the initial network. This is the network that has a variable such that its classification should be improved by the algorithm.

The experiments built for testing BBSL against the other algorithms used a combination of the several factors presented before that could affect execution. Others could be considered, but the more factors to consider the more it rapidly increases the number of experiments to be done. The experiments were a combination of target networks (ALARM, Child, Hailfinder and Insurance), initial networks (the networks generated by random local changes applied), number of examples in the dataset and percentual of missing data in examples. In a total of $4 \cdot 5 \cdot 2 \cdot 4 = 80$ configurations, for each algorithm. If you consider the 4 algorithms being run, you have a total of 320 configurations. Not enough, to try to reduce the variance in the results obtained, 10-fold-cross-validation was employed, resulting in 3200 executions of algorithms per try.

MMHC and GS algorithms have the restriction of starting from totally disconnected networks. This was done in the experiments. In the case of MMHC and GS, five different parameter initialization sets were considered. Adaptations were needed in the sense of using constraint-based algorithms with incomplete data. As cited before, the ESS were computed before execution for use in the independence and conditional independence tests.

The mean from the results of the 10-fold-cross-validation was computed and from these means, other were taken through configurations of the experiments to present them at the results in the tables. Also, statistical tests were realized (paired t-tests) to determine if the difference between the values found in the experiments were significant.

At all the experiments the parameters used for BBSL are: a) classification threshold, used to determine if the class variable was classified correctly or not, equal to 0.7; b) minimal occurrence percentual of the variable in the counts from the results of Bayes Ball algorithm, necessary so that the variable is not eliminated in the first step of BBSL, equal to 10%; c) SEM convergence threshold, to determine if SEM has already converged, equal to 0.01.

These parameters were obtained from preliminary tests, trying to achieve best results. The classification threshold was determined in order to restrict results to high quality classifications, over 70%. Minimal occurrence percentual was selected in order to avoid a small number of examples to force the inclusion of more variables in the RelevantVariables set. This way, changes in the network are done in order to achieve greater improvements in the classification score used, not just to improve the classification for a small set of examples. As EM uses to converge rapidly, with high increase of the likelihood score, a limit of maximum three iterations per EM execution was selected.

To check if differences in the presented results are statistically significative, paired t-tests were realized for each case presented in the tables, for all four algorithm comparisons. Analyzing the p-value for each test, comparing BBSL to each one of the four algorithms, results allowed us to define with high significance levels (higher than 95 %), that the differences are statistically significative, for most of the experiments. This can be seen, as the results are shown in the Appendix of this work.

More detailed tables of results are also shown in the appendix of this work.

4.4 Results

In this section, results and discussion are presented considering the experiments made the way described before. As expected of any good machine learning approach, the results shown refer to the test set. In this specific case, as said before, these results were obtained using 10-fold-cross-validation. The results presented in the tables are the average of the results obtained for each fold used in the tests, not used in the training. Compared algorithms are BBSL, DAHVI, GS, MMHC and SEM_{GHC}.

Everytime a score was neeeded to evaluate and compare candidate networks, to be fairer, the score used was the AIC_{CLL} score. It was like this for BBSL, DAHVI and SEM_{GHC} that make comparisons based on score evaluations.

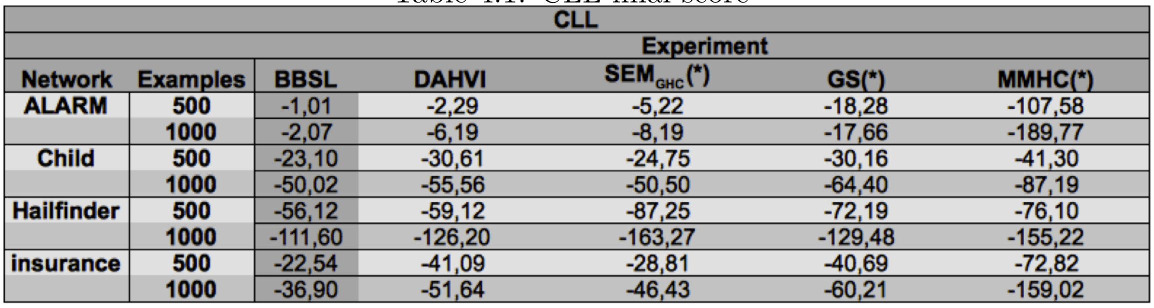
Henceforth, in the tables that will be presented, darkened results in a collumn correspond to the best results found in the comparisons with the other algorithms in the table. If the score should be maximized it corresponds to the maximum value in the line of the table, however, if the score should be minimized, the best value, darkened, will be the minimum value in the line. If a value is underlined in the table, it means that, although it may be the best result, the difference to the other algorithms is not statistically significative for at least one other algorithm in the same table.

4.4.1 CLL

The first results to be presented here are the CLL score results, since the proposed algorithm tries to improve the classification of a variable. As shown in Section 3.3, CLL presents overfitting problems related to using it when comparing Bayesian networks. That means, when using CLL to evaluate which of a number of Bayesian network structures has higher CLL score during search, the result will usually be a more connected network, because as shown in Section 3.3, this score is commonly better for structures with more edges between variables. This behaviour tends to cause overfitting problems. Because of this, in the BBSL algorithm comparisons, AIC_{CLL} was adopted for optimization. This way, the classification of the variable is improved, as shown by the CLL score increase.

The results obtained for the networks learned with BBSL show that the average CLL (Conditional Log-Likelihood) scores obtained from the test set are good in the comparisons, not presenting the overfitting problem (the score used for optimization was AIC_{CLL}). This can be considered an evidence that AIC_{CLL} works well for the comparisons between candidate Bayesian networks, when trying to improve the classification of a variable. The average results for the CLL are presented in Table 4.1 (the greater the result the best).

Table 4.1: CLL final score

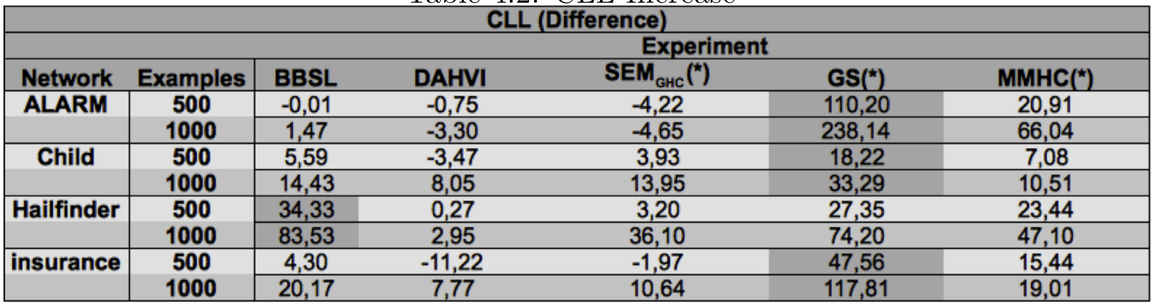


CLL						
		Experiment				
Network	**Examples**	**BBSL**	**DAHVI**	**SEM$_{GHC}$(*)**	**GS(*)**	**MMHC(*)**
ALARM	**500**	-1,01	-2,29	-5,22	-18,28	-107,58
	1000	-2,07	-6,19	-8,19	-17,66	-189,77
Child	**500**	-23,10	-30,61	-24,75	-30,16	-41,30
	1000	-50,02	-55,56	-50,50	-64,40	-87,19
Hailfinder	**500**	-56,12	-59,12	-87,25	-72,19	-76,10
	1000	-111,60	-126,20	-163,27	-129,48	-155,22
insurance	**500**	-22,54	-41,09	-28,81	-40,69	-72,82
	1000	-36,90	-51,64	-46,43	-60,21	-159,02

The increase in the CLL score from the initial network to the network learned by the algorithm is presented in Table 4.2. The results show us that indeed the final CLL for some networks were good, the initial networks already had a good score, such that the algorithm in some cases, like in the average for the ALARM network results has not improved the score very much. However, it also shows us that BBSL increased the score more than SEM_{GHC} and DAHVI, that started from the same point, resulting in a better scored network. Negative values mean that the CLL for the learned network is worse than the CLL of the initial network.

One important fact to notice when analysing these results is that GS and MMHC start from a disconnected network because of own algorithms restrictions. For BBSL,

Table 4.2: CLL Increase



CLL (Difference)							
		Experiment					
Network	Examples	BBSL	DAHVI	SEM_{GHC}(*)	GS(*)	MMHC(*)
ALARM	500	-0,01	-0,75	-4,22	110,20	20,91
	1000	1,47	-3,30	-4,65	238,14	66,04
Child	500	5,59	-3,47	3,93	18,22	7,08
	1000	14,43	8,05	13,95	33,29	10,51
Hailfinder	500	34,33	0,27	3,20	27,35	23,44
	1000	83,53	2,95	36,10	74,20	47,10
insurance	500	4,30	-11,22	-1,97	47,56	15,44
	1000	20,17	7,77	10,64	117,81	19,01

SEM_{GHC} and DAHVI the simulated expert starting point was considered. Continuing, it is reasonable to consider, that starting from a disconnected network, allows a bigger improvement in the score. This would happen because the score of the initial structure is more distant from the ideal situation, that means, from the score of the structure that generated the data. As the simulated expert structure changes were made only to a local neighbourhood of the network that generated the data, to improve this network very much is harder then to improve a very different one. This seems to justify the better increases for GS and MMHC, although the final results were not the best, even for GS that were next to the second best result.

Detailed results for the CLL are presented in Table 4.3. Notice in the table that some results are unavailable due to problems during algorithm execution, an operational issue. Other results are not in the table because they haven't finished execution in less than eight hours per fold, these are indicated as "Timeout".

For the results, notice that BBSL has the best for most of all the algorithms presented. However, also notice that except for Hailfinder (the biggest network considered), results are not statistically significative for 95% of confidence. Anyway, the CLL of the learned network is better or as good as the others when using BBSL, for all the cases presented. As it will be shown, the execution time is usually shorter for BBSL than for the other algorithms used in the comparison.

4.4.2 Time

It is important to notice for the experiments, that given incomplete data, the execution time tends to increase with the number of examples and percentual of missing data. This time is greater than the one consumed for the execution of the cited algorithm with complete data, usually presented in the literature. In our experiments

Table 4.3: Detailed Average CLL

CLL							
			Experiment				
Network	Examples	Missing	BBSL	DAHVI	SEM_{GHC}(*)	GS(*)	MMHC(*)
ALARM	500	1%	-1,64	Not Available	-9,33	-12,34	-118,24
		5%	-0,77	Not Available	-4,53	-22,88	-129,21
		15%	-1,23	-2,27	-4,11	-22,55	-84,58
		30%	-0,38	-2,31	-2,90	-15,35	-98,29
	1000	1%	-2,61	Not Available	-9,39	-16,59	-200,52
		5%	-1,42	Not Available	-7,91	-17,69	-201,07
		15%	-1,40	-8,37	-5,49	-9,77	-237,06
		30%	-2,86	-4,00	-9,98	-26,60	-120,44
Child	500	1%	-21,51	-28,49	-23,93	-33,88	-41,95
		5%	-25,18	-35,20	-25,35	-30,12	-42,92
		15%	-22,56	-26,51	-24,66	-30,54	-40,25
		30%	-23,15	Timeout	-25,07	-26,10	-40,09
	1000	1%	-52,53	-58,69	-52,82	-60,02	-92,18
		5%	-49,41	-53,56	-50,16	-62,65	-89,41
		15%	-47,05	-51,31	-47,43	-83,35	-86,10
		30%	-51,08	Timeout	-51,58	-51,60	-81,04
Hailfinder	500	1%	-56,36	-60,09	-98,83	-77,02	-84,34
		5%	-56,98	-58,25	-86,33	-63,99	-70,79
		15%	-55,70	-57,12	-87,82	-77,32	-81,12
		30%	-55,44	-64,76	-76,03	-70,44	-68,15
	1000	1%	-112,00	-121,65	-158,98	-123,97	-170,94
		5%	-111,50	-130,25	-152,17	-135,34	-165,10
		15%	-111,10	-129,38	-167,70	-130,81	-129,61
		30%	-111,79	-128,08	-176,98	-127,78	Not Available
insurance	500	1%	-25,68	-40,29	-33,63	-40,08	-62,16
		5%	-21,73	-41,89	-27,63	-49,44	-89,15
		15%	-21,81	Timeout	-26,23	-35,68	-71,08
		30%	-20,95	Timeout	-27,75	-37,59	-68,87
	1000	1%	-37,60	-51,64	-43,77	-50,35	-177,60
		5%	-37,74	Timeout	-44,68	-80,90	-137,32
		15%	-36,69	Timeout	-46,02	-47,24	-178,78
		30%	-35,56	Timeout	-51,25	-62,37	-142,37

with BBSL, this is presented in Figure 4.1.

Notice, that in these charts, sometimes the execution time when 1% of missing values is used is greater than the execution time for 5%. This is not the expected behaviour in structure learning algorithms. However, this could be explained by the fact that for different missing values in an example different sets are returned by the Bayes Ball algorithm, and so by the CollectRelevantVariables step. As different sets are returned, the number of candidate structures may differ, and then the time spent to compare structures. Since the difference in the number of examples with missing values for 1% and 5% in datasets with 500 and 1000 examples, is not so big if compared to the other percentuals presented, this effect might be the cause of the results presented.

In terms of execution time, BBSL was the fastest for most of the cases, taking more time for Insurance network. The better results are probably related to the local approach used, since GS, MMHC, and SEM_{GHC}, use global approaches. The results for DAHVI may be justified by the optimizations done in BBSL, like filtering the size of the set of variables that enter the SelectStructure step by considering the

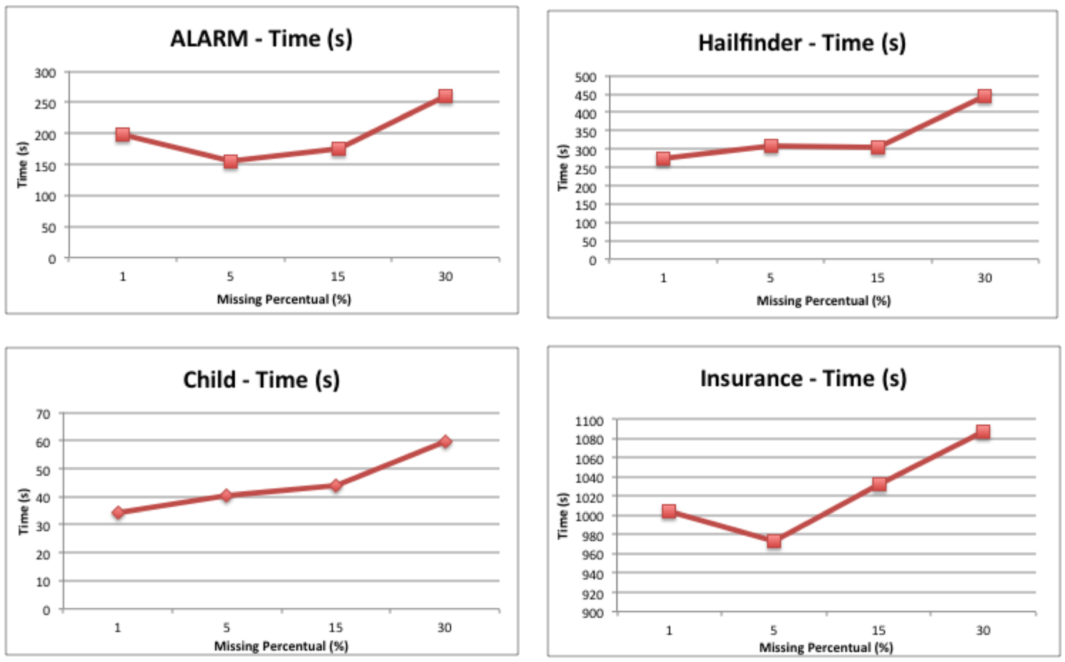


Figure 4.1: BBSL - Total execution time (seconds)

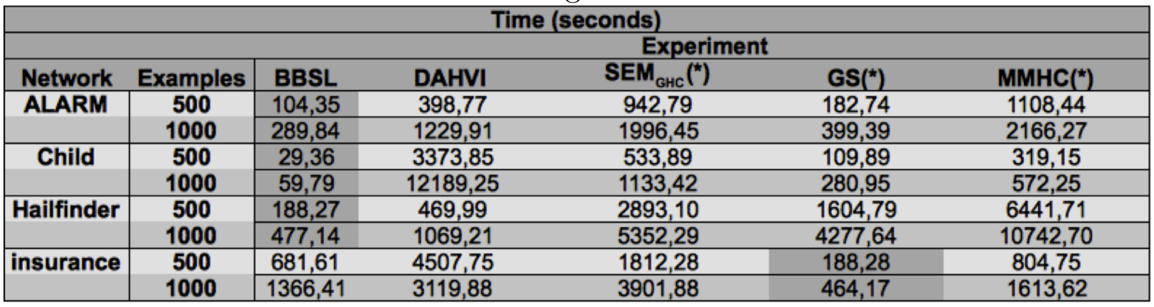
percentual of examples that could be affected by the inclusion of new variables in the search. Other optimization issue is related to the fact that DAHVI runs EM for every candidate network, what BBSL does not, since EM expectation step consumes too much computational resources BBSL can achieve better results. Both GS and MMHC, in its original publications, does not allow searching just a piece of the network at the same time it returns all the structure of a Bayesian network, and were considered this way [4] [7].

SEM_{GHC} would search locally only if a variant was used, like it is done in the second step of BBSL, restricting the search for only some networks. The networks that makes changes only to the surroundings of the class variable, and the definition of surroundings would depend on the algorithm proposed (like the Markov Blanket or the BBSL CollectRelevantVariables set). Then, it would be another algorithm that could be proposed, not exactly GHC.

One possible explanation to the execution time results for BBSL in Insurance network, is that, the "SocioEcon" variable, considered as the classification variable in the experiments is very connected. Its degree is 8, resulting in a bigger BBSL CollectRelevantVariables set. Then, with more variables returned by CollectRelevantVariables, more possible candidates are analyzed and the time consumption increases.

When considering the Hailfinder results, the network with the greatest number of variables (56), notice that the difference in time is greater than usual. This is an evidence that BBSL could be used in more big-sized networks, without great damages to execution time. In this case, for the approach that was adopted, using not so big networks, network connectivity and missing values have greater relative impact than it would be if the size of the network was bigger. Even if results were worse, differences in the CLL score for the same network, in some situations, may not justify the additional time used, favoring BBSL instead of other algorithms.

Table 4.4: Average execution time



Time (seconds)						
		Experiment				
Network	**Examples**	**BBSL**	**DAHVI**	**SEM_{GHC}(*)**	**GS(*)**	**MMHC(*)**
ALARM	**500**	104,35	398,77	942,79	182,74	1108,44
	1000	289,84	1229,91	1996,45	399,39	2166,27
Child	**500**	29,36	3373,85	533,89	109,89	319,15
	1000	59,79	12189,25	1133,42	280,95	572,25
Hailfinder	**500**	188,27	469,99	2893,10	1604,79	6441,71
	1000	477,14	1069,21	5352,29	4277,64	10742,70
insurance	**500**	681,61	4507,75	1812,28	188,28	804,75
	1000	1366,41	3119,88	3901,88	464,17	1613,62

The paired t-tests results show that these execution times that are usually better for BBSL than for other algorithms, are also statistically significative. There are only a few exceptions as shown in Table 4.5. These results confirm that BBSL results are achieved faster with equal or better quality than the algorithms considered in the comparisons.

4.4.3 AIC_{CLL}

Results in Table 4.6 use the AIC_{CLL} score. AIC_{CLL} is the optimized score in BBSL, it was used when comparing Bayesian networks in the SelectStructure step. Results presented are the mean of the increase of the score from the initial structure to the learned network using the algorithm. In the mean calculation, the several folds executed were considered, the several initial structures and the percentuals of incomplete data, for 500 and for 1000 examples, at each of the networks (ALARM, Child, Hailfinder, Insurance).

Remember that for AIC score the lesser the value obtained the better. At all results, considering only GS and MMHC, the BBSL algorithm presented a value of AIC_{CLL} better than the other two algorithms. However, when the comparison

Table 4.5: Detailed Average Time Spent (in seconds)

Time (seconds)							
			Experiment				
Network	Examples	Missing	BBSL	DAHVI	SEM_{GHC}(*)	GS(*)	MMHC(*)
ALARM	500	1%	83,31	Not Available	783,57	132,28	797,18
		5%	95,06	Not Available	871,23	148,70	744,93
		15%	<u>109,22</u>	139,17	1040,95	179,67	1157,01
		30%	<u>129,80</u>	658,37	1075,43	270,33	1734,61
	1000	1%	311,07	Not Available	1868,77	273,40	1728,60
		5%	216,16	Not Available	1596,80	280,00	1627,35
		15%	243,83	1944,29	1797,30	407,13	2047,47
		30%	<u>388,30</u>	515,53	2722,95	637,01	3261,67
Child	500	1%	20,74	189,34	431,53	80,59	251,54
		5%	24,41	568,54	503,88	84,78	243,22
		15%	28,46	13356,88	532,10	112,27	308,26
		30%	43,82	Timeout	668,04	161,92	473,56
	1000	1%	47,65	9612,06	890,77	206,06	387,04
		5%	56,13	14838,07	999,01	227,00	448,24
		15%	59,54	14694,82	1116,76	281,39	549,32
		30%	75,83	Timeout	1527,14	409,35	904,42
Hailfinder	500	1%	160,55	423,09	2207,00	1153,45	4344,65
		5%	172,23	440,13	2378,38	1195,13	5136,16
		15%	188,79	553,12	3000,88	1485,11	6003,49
		30%	231,50	450,88	3986,14	2585,49	10282,55
	1000	1%	388,31	1317,35	4193,74	3055,54	9442,60
		5%	442,55	601,70	4672,81	3296,93	10013,03
		15%	419,85	800,39	5897,59	4200,56	12772,47
		30%	657,84	1309,26	6968,22	6557,53	Not Available
insurance	500	1%	684,92	5632,35	1617,82	132,72	585,73
		5%	614,18	3383,15	1620,49	148,93	616,66
		15%	707,24	Timeout	1815,50	192,66	832,14
		30%	720,11	Timeout	2195,30	278,80	1184,46
	1000	1%	1323,12	3119,88	3226,33	321,82	752,88
		5%	1331,22	Timeout	3487,06	345,78	1314,55
		15%	1356,44	Timeout	3845,43	430,95	1714,30
		30%	1454,84	Timeout	5048,72	758,10	2672,74

was made to the SEM_{GHC}, results were worse at all cases. The GHC variant considered, despite of having a reduced number of networks to compare per iteration, increases the AIC_{CLL} score at each iteration, using a greedy heuristic. This heuristic considers modifications at any edge of the Bayesian network, so that the number of possible modifications available when trying to improve the score is greater. In BBSL, changes are only allowed to the edges that connect variables in the CollectRelevantVariables set obtained. However, the time consumed in the search even for the reduced number of network comparisons is is still high for SEM_{GHC}, as seen in Table 4.4

The average AIC_{CLL} score for the learned Bayesian networks produced in the experiments are presented in Figure 4.7.

Notice that, in Table 4.7 and in Table 4.6, indeed the score increase shows good results in the comparisons, the score of the final learned network is the worst among all the algorithms. One possible cause for this is that algorithms like MMHC are delivering highly disconnected networks, therefore with a reduced number of

Table 4.6: AIC_{CLL} Increase

AIC_{CLL} (Difference)						
		Experiment				
Network	**Examples**	**BBSL**	**DAHVI**	**SEM_{GHC}(*)**	**GS(*)**	**MMHC(*)**
ALARM	**500**	-236,31	-110,55	-956,91	31,39	23,49
	1000	-453,40	-123,33	-906,95	-66,84	-65,22
Child	**500**	-119,17	-37,05	-640,88	200,56	16,64
	1000	-140,15	-100,75	-646,83	432,82	0,49
Hailfinder	**500**	-1978,21	-306,78	-3191,57	837,90	231,82
	1000	-2305,91	-627,97	-2989,42	1192,40	175,80
insurance	**500**	-556,59	-143,87	-3109,43	288,58	42,42
	1000	-502,96	-43,68	-3077,89	366,63	26,24

Table 4.7: AIC_{CLL} final score

AIC_{CLL}						
		Experiment				
Network	**Examples**	**BBSL**	**DAHVI**	**SEM_{GHC}(*)**	**GS(*)**	**MMHC(*)**
ALARM	**500**	1579,69	1706,54	859,08	498,36	490,46
	1000	1367,69	1758,61	914,13	654,78	656,39
Child	**500**	782,20	864,15	260,49	417,32	233,40
	1000	832,74	873,23	326,07	748,21	315,87
Hailfinder	**500**	8470,29	9354,79	7256,93	1482,98	876,90
	1000	8351,95	8459,52	7448,80	2045,76	1026,43
insurance	**500**	3164,28	3550,38	611,45	643,09	396,93
	1000	3278,37	3423,15	703,45	900,68	560,28

parameters. In this way, the penalty value of Akaike for these networks is too small, what highly decreases the score. Possibly, using AIC_{CLL} is giving a higher weight to the number of parameters than it should do. Notice however, that when comparing candidate networks, the structures being compared differ only from local changes, such that they are similar in the edges that are away from the local surroundings. Then, the number of parameters of these networks are more similar, and the choice is more dependent of the CLL score, still penalizing complexity.

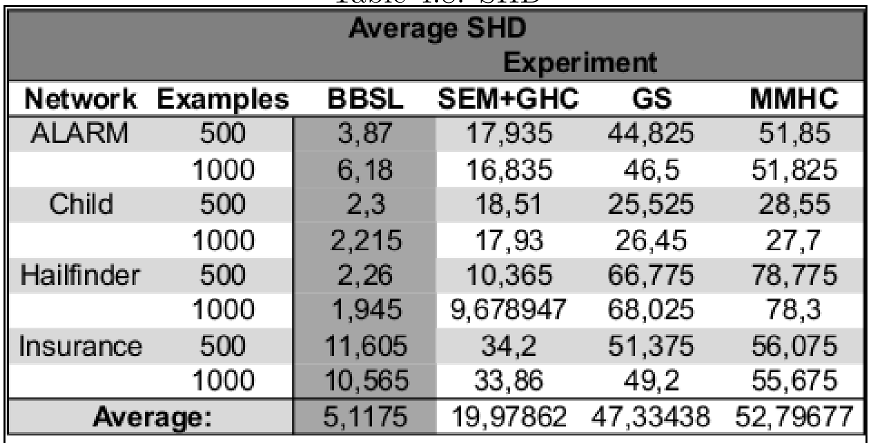
From the results that have been presented untill here, it may be said that AIC_{CLL} score can be considered useful to compare structures in order to try to improve the CLL of the resulting Bayesian network. Also, that this CLL improvement is greater than the SEM_{GHC} improvements, that started at the same conditions.

As it will be presented on Table 4.11, in the case where BBSL starts from a network similar to the ones GS and MMHC start, results are comparable to GS ones in the average. On Table 4.10 where the start is random, and the nodes are already connected, in such a way BBSL can take advantage of the existent connections for Bayes Ball execution, the results are even better.

4.4.4 Structural Hamming Distance (SHD)

Table 4.8 presents the Structural Hamming Distance (SHD) [7]. It is defined by the number of operations (additions, removals and inversions of edges) that would be necessary to transform one structure into another. The comparisons were made between the resulting learned networks and the target networks. These last are the original networks from which the datasets were sampled.

Table 4.8: SHD



Average SHD					
		Experiment			
Network	Examples	BBSL	SEM+GHC	GS	MMHC
ALARM	500	3,87	17,935	44,825	51,85
	1000	6,18	16,835	46,5	51,825
Child	500	2,3	18,51	25,525	28,55
	1000	2,215	17,93	26,45	27,7
Hailfinder	500	2,26	10,365	66,775	78,775
	1000	1,945	9,678947	68,025	78,3
Insurance	500	11,605	34,2	51,375	56,075
	1000	10,565	33,86	49,2	55,675
Average:		5,1175	19,97862	47,33438	52,79677

SHD results in Table 4.8 had smaller values for BBSL, what indicates that the obtained network differs less from the gold model, the target structure. Evaluating the results for GS and MMHC, it may be said that in part this is caused by the differences in initial networks. BBSL and SEM_{GHC} already started from a structure that is not very different from the target one. DAHVI is not presented because it inserts hidden variables, so the number of variables is also not equal to the ones in the other structures.

4.4.5 Accuracy

In the Figure 4.9, the accuracy of the final resulting learned network is compared among the algorithms. Results show that in the end of the learning task, BBSL still has a good percentual of accuracy.

Table 4.9: Accuracy

Accuracy						
		Experiment				
Network	Examples	BBSL	DAHVI	SEM_{GHC}(*)	GS(*)	MMHC(*)
ALARM	500	0,991	0,992	0,981	0,905	0,174
	1000	0,989	0,980	0,986	0,951	0,286
Child	500	0,798	0,793	0,799	0,798	0,719
	1000	0,800	0,806	0,802	0,783	0,704
Hailfinder	500	0,075	0,060	0,142	0,172	0,038
	1000	0,152	0,028	0,203	0,164	0,047
insurance	500	0,802	0,820	0,818	0,663	0,003
	1000	0,828	0,850	0,840	0,763	0,000

4.5 BBSL Configurations Analysis

In order to try to achieve better results, some other configurations of BBSL were tested. In this section, some results are presented for a part of these approaches. Among the possible configurations the ones selected were:

- (I) Starting BBSL from a randomly generated Bayesian network, as if there were not any background knowledge about the domain (results in Table 4.10);
- (II) Starting BBSL from a totally disconnected Bayesian network, where there are no edges, as if all nodes are independent of all the others. In this case, the External Edges operator was used, in order to solve the problem of the bayes ball set of the node being empty when the class variable is independent of all other nodes (results in Table 4.11);
- (III) Starting BBSL using the network that generated the data with a local modification on its structure in the neighbourhood of the class variable, simulating a network structure elicitated by an expert (results in Table 4.12);
- (IV) Starting BBSL using the network that generated the data with a local modification on its structure in the neighbourhood of the class variable, simulating a network structure elicitated by an expert and using the External Edges operator (results in Table 4.13).

Table 4.10: BBSL - Randomly Initialized

BBSL - Randomly Initialized							
Network	Examples	AIC+CLL Final	AIC+CLL Diff	CLL Final	CLL Diff	SHD	Time (s)
ALARM	500	4957,67	-952,35	-16,81	155,60	123,98	468,74
	1000	4914,46	-1339,64	-28,97	315,49	123,38	1249,41
Child	500	1303,43	-187,95	-31,97	35,09	45,93	113,20
	1000	1373,61	-253,01	-66,62	68,06	46,07	241,60
Hailfinder	500	56561,99	-6530,75	-58,71	78,76	212,25	5824,22
	1000	51269,02	-3120,65	-115,14	163,69	211,49	8001,40
Insurance	500	4176,01	-1149,97	-44,61	81,38	89,63	571,70
	1000	4374,50	-1328,36	-81,70	172,73	89,89	1303,07
Average:		16116,34	-1857,84	-55,57	133,85	117,83	2221,67

Table 4.11: BBSL - Started from Disconnected Network

BBSL - Started from Disconnected Network							
Network	Examples	AIC+CLL Final	AIC+CLL Diff	CLL Final	CLL Diff	SHD	Time (s)
ALARM	500	269,41	-197,56	-27,91	100,58	46,20	45,86
	1000	322,89	-398,72	-52,67	203,14	46,33	100,02
Child	500	198,20	-18,57	-36,40	11,98	25,48	22,46
	1000	283,37	-32,02	-75,96	21,73	25,50	44,28
Hailfinder	500	574,64	-70,45	-64,32	35,22	66,00	72,71
	1000	708,75	-144,60	-128,98	74,70	66,25	141,96
Insurance	500	290,83	-63,68	-54,09	34,16	52,23	35,76
	1000	407,33	-126,71	-108,07	69,96	52,50	67,93
Average:		381,93	-131,54	-68,55	68,93	47,56	66,37

Table 4.12: BBSL - Started from Simulated Expert Network

BBSL							
Network	Examples	AIC+CLL Final	AIC+CLL Diff	CLL Final	CLL Diff	SHD	Time (s)
ALARM	500	1579,69	-236,31	-1,01	-0,01	3,87	104,35
	1000	1367,69	-453,40	-2,07	1,47	6,18	289,84
Child	500	782,20	-119,17	-23,10	5,59	2,30	29,36
	1000	832,74	-140,15	-50,02	14,43	2,22	59,79
Hailfinder	500	8470,29	-1978,21	-56,12	34,33	2,26	188,27
	1000	8351,95	-2305,91	-111,60	83,53	1,95	477,14
Insurance	500	3164,28	-556,59	-22,54	4,30	11,61	681,61
	1000	3278,37	-502,96	-36,90	20,17	10,57	1366,41
Average:		3478,40	-786,59	-37,92	20,48	5,12	399,60

Table 4.13: BBSL - Started from Simulated Expert Network with External Edges

BBSL - External Edges							
Network	Examples	AIC+CLL Final	AIC+CLL Diff	CLL Final	CLL Diff	SHD	Time (s)
Alarm	500	1215,76	-600,23	-2,72	-1,72	8,10	373,42
	1000	1210,32	-610,76	-2,76	0,78	8,23	674,94
Child	500	782,20	-119,17	-23,10	5,59	2,30	62,02
	1000	832,74	-140,15	-50,02	14,43	2,22	112,15
Hailfinder	500	8991,17	-1457,33	-56,57	33,88	2,35	558,21
	1000	8996,90	-1660,96	-111,92	83,21	2,20	1149,53
Insurance	500	-	-	-	-	-	-
	1000	-	-	-	-	-	-
Average:		3671,51	-764,77	-41,18	22,69	4,23	488,38

Notice that results for the Insurance network in Table 4.13 are not available due to operational problems.

If comparing the final CLL of the network is the criterion adopted, the best configuration would be (III) considering the average. If the CLL increase is the selected parameter of comparison, the randomly initialization would be the best choice (I) followed by (II), probably explained by the fact that there is more opportunity to improve a structure that is farther from the ideal one. The same may explain a better SHD in (III) and (IV). The AIC_{CLL} score is worst for (I), probably because of the number of parameters, not high in (II).

Also notice that, using the External Edges operator in BBSL (compare III and IV), makes BBSL more dependent on the size of the network, increasing the time spent. The bigger the network, more time will be spent when using External Edges because edges are added connecting the class variable to each of the other variables. This behaviour produces an addition of a number of candidate networks to be evaluated that is twice the number of variables excluding the class variable.

Chapter 5

Related Work

There are several algorithms in the literature that are in some way related to the algorithm developed in this work. They are structure learning algorithms that have one or more characteristics in common with BBSL. In this chapter, these works are briefly presented, and differences between them and BBSL are pointed out.

There are already several algorithms to learn Bayesian network structure in the literature. They are divided into three main types as presented earlier: constraint-based, score-based or hybrid algorithms. Algorithms at each of these three classes are considered in this chapter.

Some of the algorithms that use a constraint-based approach include GS(Grow and Shrink)[4] [42], SGS [43], KS (Koller-Sahami) [44], PC [45], IAMB (Incremental Association Markov Blanket) [46] and its variations like Inter-IAMB [46], InterIAMBnPC [46], λ-IAMB [47], HITON [48], Fast-IAMB [49]. Despite of being constraint-based, not all of these algorithms try to learn all the structure of the Bayesian network.

Some of the algorithms cited above only try to discover the Markov Blanket of a given variable, an important task, since researchers have suggested that the Markov Blanket is a key concept to variable selection [48]. However, this behaviour can be used to learn the structure of the network. This is the case of GS as referred in [4] where it is separated into GS Markov Blanket (GSMB) and GS, the second is used to learn all the structure of the network using the first. Something similar happens to PC, where sometimes its version for learning the MB is considered, and other times its full version to learn all the Bayesian networks with oriented edges like in

[45].

Algorithms like IAMB, a two phase algorithm like GS, that is not very different from GS except for the dynamic ordering of the variables during the growing phase, only learn the MB [46]. Its variations like Inter-IAMB, InterIAMBnPC and the more recent λ-IAMB also only learn the MB [46] [47]. λ-IAMB is a slight variation of IAMB with computer time gains in conditional mutual information calculation, since it decomposes it in function of entropy and computes only the factors that change from one independence test to another [47]. Because independence tests are done repeated times during a constraint-based algorithm execution and are considered a common measure of performance for these algorithms, enhancing independence test performance is a good upgrade.

More recent algorithms include HITON, Fast-IAMB and λ-IAMB. All of them learn the Markov Blanket of the node [48] [49] [47]. HITON has a different approach from the other two since it learns the parent and children set of the variable first, then make the union of this set to the parents and children set of each parent and children of the node and tries to identify, in another step, the spouses and nodes that does not pertain to the MB set of the variable [48]. Fast-IAMB and λ-IAMB have in common that both are variations of the IAMB algorithm [49] [47].

KS (Koller-Sahami) algorithm is not dedicated to discovering the MB of a variable or even to learn Bayesian structure, but it is useful to make feature selection, and accepts parameters that when the size of the MB is known allows the user to learn the MB of the variable and so it is used in comparisons in some works like [48] [46].

Notice that some of these algorithms have variants that were not explicited here. Also there are other works in the same orientation.

Some examples of score-based algorithms are GHC (Greedy Hill-Climbing) [3], Simmulated Annealing [9] (and variants like TLSA [50] and Parallel TLSA [50]), K2 [16], GES (Greedy Equivalent Search) [51], Sparse Candidate [6], Optimal Reinsertion [5], B&B (Branch & Bound) [11], DBMC (Dynamic Markov Blanket Classifier) [10]. From these, all of them use its respective heuristics and scores to search the Bayesian networks space of solutions trying to solve the optimization problem of learning the structure. DMBC [10] is a local algorithm that restricts its search to the Markov Blanket of the network, improving classification locally. B&B [11] tries to apply combinatorial optimization techniques to improve the score of the learned network.

Hybrid algorithms include MMHC [7] and the more recent H2PC [17]. Both use a constraint-based approach in a previous step to restrict the search space of possible solutions, and then in a second step use a score-based heuristic to navigate the search space while optimizing the solution. In this concern, BBSL behaves the same way, with two steps, one for restricting the space of solutions and the other for searching and optimizing. One of the differences comes from the fact that BBSL does not use conditional independence tests to restrict the search space, but the d-separation concept. Other difference is that BBSL only searches a local neighbourhood, determined by its first step using the Bayes Ball algorithm, improving an existent network locally, instead of starting from a totally disconnected structure. This way, BBSL can use the existent background knowledge for the domain. Also, BBSL is more example oriented than the other two, trying to improve the network for most of the examples for which classification does not perform well. Besides this, BBSL focus on the classification problem, in what it differs from MMHC and H2PC.

When considering local structure learning algorithms, there are, at least, SLL, DMBC and DAHVI.

SLL [19] is a local score-based search algorithm, with a global variation called SLL+G and a constraint-based variation called SLL+C. SLL executes in a more comparable time to a constraint-based algorithm by using a local approach. It is still worst when considering time consumption but it is already a competitive alternative to constraint-based algorithms [19]. Usually score-based alternatives are slower than constraint-based ones and SLL approach tries to be an intermediary option [19]. It differs from BBSL in some important aspects, first its local search considers the Markov Blanket of the node. In BBSL a more comprehensive concept, the d-separation, is considered, as explained in the introduction. Other important difference is that SLL is not adequate for incomplete data, because its sub-routine OptimalNetwork uses an algorithm that has this restriction. SLL is also not focused on the classification problem.

Other alternative for local learning cited is the DMBC[10] algorithm. This algorithm looks for the optimal solution to the structure search problem in the Bayesian network restricting itself to the Markov Blanket of the variable [10]. BBSL as cited before uses the d-separation concept to choose the set of variables that affect the probability distribution of the variable, being more comprehensive, and is able to consider statistical relationships existent between variables in incomplete datasets that would not exist for complete examples. DMBC, like BBSL, is focused on the classification problem. Also, DMBC is not suitable for incomplete data problems since the K2 score used, has factorials that cannot be calculated for the decimal numbers in Expected Sufficient Statistics. And finally, DMBC, as a K2-based algorithm, needs some variable ordering before its execution.

Perhaps the most similar algorithm to BBSL is DAHVI (Discriminative Approach for Hidden Variable Introduction) [20], both are local, change the structure of the Bayesian network, deal with incomplete data and try to improve the classification of variables. However, DAHVI tries to accomplish its objective of improving the classification of a group of variables by inserting hidden variables in the current structure. As it will be shown, BBSL tries to improve the classification of only one variable by changing the edges of the structure in the surroundings of the variable. Although BBSL was designed for use with just one variable, it could be easily extended to the case of several variables classification improvement, as DAHVI does. DAHVI, as DMBC, also uses the Markov Blanket of the node, actually, it uses the referred MB* [20] set that is an extension of the MB concept, including the MBs of the variables that are missing for a specific example recursively.

The MB* set of variables differs from the set returned by the Bayes Ball algorithm, used in BBSL, as proven with a counterexample:

Consider the Bayesian network structure with variables A, B, C, D e E. This structure presents two v-structures $A \rightarrow B \leftarrow C$ and $C \rightarrow D \leftarrow E$. B and E are observed variables (they are in the evidence). A is the class variable ($P(A|B,E)$).

Using MB*, C pertains to $MarkovBlanket(A)$ and is not observed. $MarkovBlanket(C)$ includes D and E, therefore $MB* = \{A, B, C, D, E\}$.

Using d-separation, the path from C to A is active through B, because B is observed. The path through C is active because C is not observed. However, the path through D is blocked, because D is not observed, so E, in this specific case, does not influence the probability distribution of A. $P(A|B,E) = P(A|B)$ and the result of Bayes Ball does not include E.

Finally, the conclusion is that the set returned by Bayes Ball is not always the same as the one returned by MB*.

Other improvements, differences and optimizations between DAHVI and BBSL include:

- BBSL defines the candidate structures in the search step as neighbours by the application of only one operator of addition, remotion or inversion of edges (as in GHC), while DAHVI defines it by the application of only one operator of hidden variable inclusion.

- BBSL eliminates variables with little presence in the resulting sets returned by Bayes Ball, the same is not done for DAHVI using the MB*. Such that a MB* set determined by only one example, may happen to include variables in the search, increasing computation time.

- DAHVI executes EM for each candidate network, while BBSL only once per iteration, reusing the expected sufficient statistics. Since EM takes very much time to execute, it has great impact on evaluation performance.

- BBSL proposes and uses the AIC_{CLL} score for classification tasks, presented in Section 3.3.

For incomplete data with global approach, there are solutions like SEM [18] and its variations that search all the space of possible Bayesian networks, in contrast to BBSL that has a local nature. This makes BBSL faster than these algorithms, however, BBSL only makes local changes.

BBSL combines desirable characteristics of several approaches cited before. It uses a local approach, improves the classification of a chosen variable and can deal with incomplete data. It also may start from an existing network, what is not common for all hybrid and constraint-based algorithms, like MMHC and GS. Like BBSL, some score-based structure search algorithms can look for an optimized solution from a known starting network, like GHC and Simulated Annealing. This is important because allows us to use existent background knowledge.

In Table 5.1, a set of characteristics of the main algorithms presented in this chapter is shown.

BBSL is a fast algorithm, competitive to some constraint-based solutions like Grow and Shrink. It also learns a Bayesian network with structure and parameters, such that it allows the user to make inference from this network. Constraint-based methods, and methods that try to learn the MB of the network sometimes only return a Bayesian network structure without parameters or a set of variables that compose the MB, without differing from parents, children and spouses. These include GS, PC and IAMB.

Table 5.1: Comparative - Algorithms Characteristics

	BBSL	DAHVI	DMBC	SEM	MMHC	GS	PC	IAMB	HITON	GHC	SA	K2	SLL
Local	o	o	o	x	x	x	x	o	o	x	x	x	o
Global	x	x	x	o	o	o	o	x	x	o	o	o	o
Incomplete Data	o	o	x	o	x	x	x	x	x	x	x	x	x
Complete Data	o	o	o	o	o	o	o	o	o	o	o	o	o
Focus on Classification Problem	o	o	o	x	x	x	x	x	x	x	x	x	x
Learns Structure and Parameters	o	o	o	o	o	x	x	x	x	o	o	o	o
Can use Background Knowledge	o	o	x	o	x	x	x	x	x	o	o	x	x

Chapter 6

Conclusions

In this final chapter, the main and general conclusions of this work are presented. Moreover, the main contributions of this work to the literature are highlighted. Also, future work indicate possible improvements and opportunities, in order to continue this research.

From what was presented at this work, the main contribution is the BBSL algorithm, an algorithm that is able to learn from incomplete data, with a local approach, that improves the classification of a variable given by the user.

BBSL uses the d-separation criterion instead of the more commonly used markov blanket of the variable. The use of the concept of d-separation for evaluating the variables that may influence the probability distribution of a random variable in the learning structure algorithm is something new to the local structure learning for Bayesian networks and could be more explored in other new algorithms.

Results have shown that BBSL consumes little time in comparison to other algorithms. It is competitive even to some constraint-based solutions. Also, it achieves its objective of acting locally to improve the classification of a variable. In this work, this was represented by the CLL score.

The score proposed intended to evaluate classifications not only by the number of examples classified correctly or not, like in accuracy, but also, how well these examples were classified. This already exists in the CLL score presented, however with AIC_{CLL} the problem of overfitting a network because of high connectivity preference is avoided.

Another topic that is worthy attention in the results, are the comparisons made.

The algorithms to which the comparisons were made do not really fit the profile of having the same proposal as BBSL. Although, they serve as good metrics to give some idea of performance of BBSL, relating its performance to the performance of other algorithms present in the literature. Adaptations made to the algorithms such that they could work with incomplete data may be seen as minor contributions, and used in future comparisons by others.

The work also complements the available literature concerning Bayesian network learning with incomplete datasets, since books usually don‘t come into detail for this topic, and literature seemed to be very fragmented. Also, the local approach applied to Bayesian network structure learning appear as a relatively recent topic.

Other issue that may come into discussion when trying to develop Bayesian network related software is the lack of software tools, libraries, and uniformity in the implementations. Every work makes comparisons to other related works, but there are no uniform implementations that can be used. Even for the algorithms that are available on the web, some are not open source, some are not even available, and there is no uniformity to the technologies used, making comparisons more dependable on the technology used, and not only on the algorithm itself. Not a small work had been employed in the Python implementations of the algorithms needed to learn structure, learn parameters, make inference, evaluate networks, make statistical tests, and so on.

BBSL allows using previous knowledge about the domain variables in its execution, a desirable characteristic not present in several algorithms that is important when this background knowledge is available. In BBSL, it can be expressed as the currently known Bayesian network, which variables are known to be statistical dependent of each one, and what is the conditional probability distribution of these variables. Using previous knowledge and being able to work with incomplete data makes BBSL more adequate to real world situations.

In what concerns execution time, the Bayesian network algorithms presented for learning structure are very sensitive to the number of variables and percentual of missing data. ESS computation is really costful and it would be very interesting to have more alternative methods. Even in this situation, BBSL executed in a reasonable time for networks with 56 variables, 30% of examples containing missing data and a thousand examples. This is possible because of the use of a local approach, since it allows that execution time doesn‘t increase so rapidly with the size of the Bayesian network.

For future work, it would be useful to solve the problem of BBSL with independent nodes. When the class variable is independent, the bayes ball algorithm only returns the class variable, so that BBSL does not make any improvement to the network strcuture, only to the parameters through EM execution. When data is complete the nodes returned from the Bayes Ball execution is the MB of the class variable.

Also it would be nice to try to build an algorithm from BBSL that uses a global approach, an algorithm that learns the edges between all pair of variables in the structure. The fact that BBSL can be run locally to the surroundings of a node, and the continuous increasing number of cores of recent computers could be used to build a parallel algorithm that could run faster even for bigger networks.

Bibliography

[1] PEARL, J., RUSSELL, S. "Bayesian networks", *The Handbook of Brain Theory and Neural Networks, 2nd edition*, 2003.

[2] DARWICHE, A. "Bayesian networks", *Communications of the ACM*, v. 53, n. 12, pp. 80–90, 2010.

[3] KOLLER, D., FRIEDMAN, N. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[4] D.MARGARITIS. *Learning Bayesian Network Model Structure from Data*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, 2003.

[5] MOORE, A., WONG, W. "Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning". In: *Proceedings of the 20th International Conference on Machine Learning (ICML 03)*, pp. 552 – 559. AAAI Press, 2003.

[6] FRIEDMAN, N., NACHMAN, I., PEÉR, D. "Learning Bayesian network structure from massive datasets: the sparse candidate algorithm". In: *Proceedings of the Fifteenth Conference on Uncertainty in artificial intelligence*, pp. 206–215. Morgan Kaufmann Publishers Inc., 1999.

[7] TSAMARDINOS, I., BROWN, L. E., ALIFERIS, C. F. "The max-min hill-climbing Bayesian network structure learning algorithm". In: *Machine Learning, Vol. 65*, pp. 31 – 78, 2006.

[8] MEEK, C. *Graphical Models: Selecting Causal and Statistical Models*. PhD thesis, Carnegie Mellon University, 1997.

[9] KIRKPATRICK, S., GELLET, C. D., VECCHI, M. P. "Optimization by simulated annealing", *Science*, pp. 671–680, 1983.

[10] DOS SANTOS, E. B., HRUSCHKA JR, E. R., HRUSCHKA, E. R., et al. "Bayesian network classifiers: Beyond classification accuracy", *Intelligent Data Analysis*, v. 15, n. 3, pp. 279–298, 2011.

[11] DE CAMPOS, C. P., JI, Q. "Efficient structure learning of Bayesian networks using constraints." *Journal of Machine Learning Research*, v. 12, n. 3, pp. 663–689, 2011.

[12] POURRET, O., NAÏM, P., MARCOT, B. *Bayesian networks: a Practical Guide to Applications*, v. 73. Wiley. com, 2008.

[13] KENETT, R. "Applications of Bayesian Networks", *Available at SSRN 2172713*, 2012.

[14] XIAO-XUAN, H., HUI, W., SHUO, W. "Using expert's knowledge to build Bayesian networks". In: *Computational Intelligence and Security Workshops, 2007. CISW 2007. International Conference on*, pp. 220–223. IEEE, 2007.

[15] HECKERMAN, D. *A Tutorial on Learning with Bayesian Networks.* Microsoft Research, 1995.

[16] COOPER, G. F., HERSKOVITS, E. "A Bayesian method for the induction of probabilistic networks from data", *Machine learning*, v. 9, n. 4, pp. 309–347, 1992.

[17] GASSE, M., AUSSEM, A., ELGHAZEL, H. "An experimental comparison of hybrid algorithms for Bayesian network structure learning". In: *Machine Learning and Knowledge Discovery in Databases*, Springer, pp. 58–73, 2012.

[18] FRIEDMAN, N. "Learning belief networks in the presence of missing values and hidden variables". In: *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 125–133. Morgan Kaufmann, 1997.

[19] NIINIMAKI, T., PARVIAINEN, P. "Local Structure Discovery in Bayesian Networks", *arXiv preprint arXiv:1210.4888*, 2012.

[20] REVOREDO, K., PAES, A., ZAVERUCHA, G., et al. "Revisando redes Bayesianas através da introdução de variáveis não-observadas", *ENIA*, 2009.

[21] KOLLER, D., FRIEDMAN, N., GETOOR, L., et al. "Graphical models in a nutshell". In: Getoor, L., Taskar, B. (Eds.), *Introduction to Statistical Relational Learning*, MIT Press, 2007.

[22] SHACHTER, R. D. “Bayes-Ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams)”. In: *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, Madison, Wisconsin*, pp. 480 – 487, 1998.

[23] ROBINSON, R. W. “Counting labeled acyclic digraphs”. In: *New Directions in the Theory of Graphs*, Academic Press, pp. 239–273, 1973.

[24] ROBINSON, R. W. “Counting unlabeled acyclic digraphs”. In: *Combinatorial mathematics V*, Springer, pp. 28–43, 1977.

[25] CHENG, J., BELL, D., LIU, W. “Learning Bayesian networks from data: An efficient approach based on information theory”, *On World Wide Web at http://www. cs. ualberta. ca/˜ jcheng/bnpc. htm*, 1998.

[26] ALIFERIS, C. F., STATNIKOV, A., TSAMARDINOS, I., et al. “Local causal and Markov blanket induction for causal discovery and feature selection for classification part i: Algorithms and empirical evaluation”, *The Journal of Machine Learning Research*, v. 11, pp. 171–234, 2010.

[27] NÄGELE, A., DEJORI, M., STETTER, M. “Bayesian substructure learning-approximate learning of very large network structures”. In: *Machine Learning: ECML 2007*, Springer, pp. 238–249, 2007.

[28] NEAPOLITAN, R. E. *Learning Bayesian Networks.* Prentice Hall, 2003.

[29] COOPER, G. F., HERSKOVITS, E. “A Bayesian method for the induction of probabilistic networks from data”, *Machine learning*, v. 9, n. 4, pp. 309–347, 1992.

[30] SCHWARZ, G. “Estimating the dimension of a model”, *The Annals of Statistics*, v. 6, n. 2, pp. 461–464, 1978.

[31] RISSANEN, J. “Modeling by shortest data description”, *Automatica*, v. 14, n. 5, pp. 465–471, 1978.

[32] COOPER, G. F., HERSKOVITS, E. “A Bayesian method for constructing Bayesian belief networks from databases”. In: *Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence*, pp. 86–94. Morgan Kaufmann Publishers Inc., 1991.

[33] MYLLYMÄKI, P., ROOS, T., SILANDER, T., et al. “Factorized NML models”, 2008.

[34] DE CAMPOS, L. M. “A scoring function for learning Bayesian networks based on mutual information and conditional independence tests”, *The Journal of Machine Learning Research*, v. 7, pp. 2149–2187, 2006.

[35] GROSSMAN, D., DOMINGOS, P. “Learning Bayesian network classifiers by maximizing conditional likelihood”. In: *In Proc. 21st International Conference on Machine Learning, Banff, Canada*, pp. 361 – 368, 2004.

[36] AKAIKE, H. “A new look at the statistical model identification”, *Automatic Control, IEEE Transactions on (Volume:19 , Issue: 6, Pages: 716-723)*, 1974.

[37] SARDINHA, R., PAES, A., ZAVERUCHA, G. “Aprendizado Local da Estrutura de Redes Bayesianas a partir de Dados Incompletos”, *X Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*, 2013.

[38] WOLFE, J., HAGHIGHI, A., KLEIN, D. “Fully distributed EM for very large datasets”. In: *Proceedings of the 25th International Conference on Machine Learning*, pp. 1184–1191. ACM, 2008.

[39] CHEN, S. F., GOODMAN, J. “An empirical study of smoothing techniques for language modeling”. In: *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, pp. 310–318. Association for Computational Linguistics, 1996.

[40] SILANDER, T., MYLLYMAKI, P. “A simple approach for finding the globally optimal Bayesian network structure”, *arXiv preprint arXiv:1206.6875*, 2012.

[41] BEINLICH, I., SUERMONDT, H. J., CHAVEZ, R. M., et al. “The ALARM monitoring system: a case study with two probabilistic inference techniques for belief networks”. In: *Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine, Springer-Verlag*, pp. 247–256, 1989.

[42] MARGARITIS, D., THRUN, S. “Bayesian network induction via local neighborhoods”. In: *Advances in Neural Information Processing Systems 12*, p. 505–511. MIT Press, 2000.

[43] SPIRTES, P., GLYMOUR, C. “An algorithm for fast recovery of sparse causal graphs”, *Social Science Computer Review*, v. 9, n. 1, pp. 62–72, 1991.

[44] KOLLER, D., SAHAMI, M. “Toward Optimal Feature Selection”. In: Saitta, L. (Ed.), *Proceedings of the Thirteenth International Conference on Machine Learning (ICML)*, pp. 284–292. Morgan Kaufmann Publishers, 1996.

[45] ABELLÁN, J., GÓMEZ-OLMEDO, M., MORAL, S. “Some Variations on the PC Algorithm.” In: *Probabilistic Graphical Models*, pp. 1–8, 2006.

[46] TSAMARDINOS, I., ALIFERIS, C. F., STATNIKOV, A. R., et al. “Algorithms for Large Scale Markov Blanket Discovery.” In: *FLAIRS Conference*, v. 2003, pp. 376–381, 2003.

[47] ZHANG, Y., ZHANG, Z., LIU, K., et al. “An improved IAMB algorithm for Markov blanket discovery”, *Journal of Computers*, v. 5, n. 11, pp. 1755–1761, 2010.

[48] ALIFERIS, C. F., TSAMARDINOS, I., STATNIKOV, A. “HITON: a novel Markov Blanket algorithm for optimal variable selection”. In: *AMIA Annual Symposium Proceedings*, v. 2003, p. 21. American Medical Informatics Association, 2003.

[49] YARAMAKALA, S., MARGARITIS, D. “Speculative Markov blanket discovery for optimal feature selection”. In: *Data mining, fifth IEEE international conference on*, pp. 4–pp. IEEE, 2005.

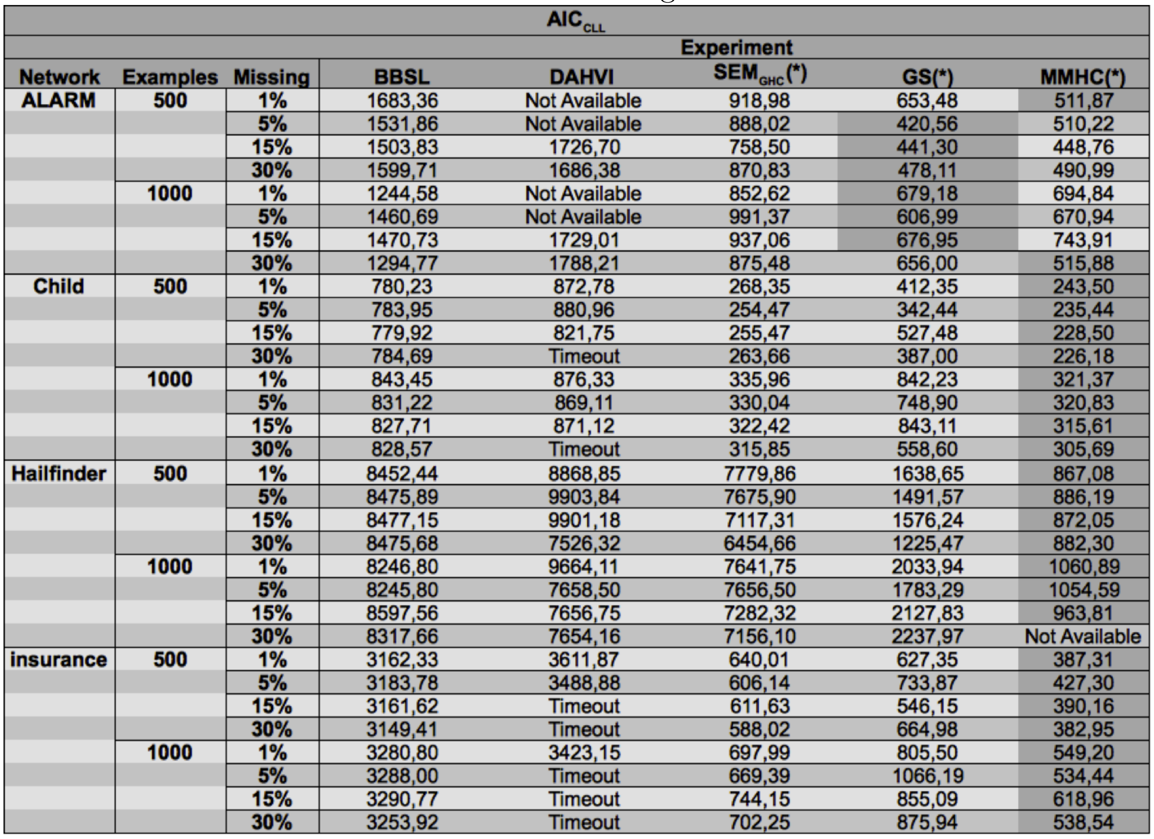
[50] XUE, G.-L. “Parallel two-level simulated annealing”. In: *Proceedings of the 7th international conference on Supercomputing*, pp. 357–366. ACM, 1993.

[51] CHICKERING, D. M. “Optimal structure identification with greedy search”, *The Journal of Machine Learning Research*, v. 3, pp. 507–554, 2003.

Appendix A

Detailed Results

Table A.1: Detailed Average AIC+CLL



					AIC$_{CLL}$		
					Experiment		
Network	**Examples**	**Missing**	**BBSL**	**DAHVI**	**SEM$_{GHC}$(*)**	**GS(*)**	**MMHC(*)**
ALARM	**500**	**1%**	1683,36	Not Available	918,98	653,48	511,87
		5%	1531,86	Not Available	888,02	420,56	510,22
		15%	1503,83	1726,70	758,50	441,30	448,76
		30%	1599,71	1686,38	870,83	478,11	490,99
	1000	**1%**	1244,58	Not Available	852,62	679,18	694,84
		5%	1460,69	Not Available	991,37	606,99	670,94
		15%	1470,73	1729,01	937,06	676,95	743,91
		30%	1294,77	1788,21	875,48	656,00	515,88
Child	**500**	**1%**	780,23	872,78	268,35	412,35	243,50
		5%	783,95	880,96	254,47	342,44	235,44
		15%	779,92	821,75	255,47	527,48	228,50
		30%	784,69	Timeout	263,66	387,00	226,18
	1000	**1%**	843,45	876,33	335,96	842,23	321,37
		5%	831,22	869,11	330,04	748,90	320,83
		15%	827,71	871,12	322,42	843,11	315,61
		30%	828,57	Timeout	315,85	558,60	305,69
Hailfinder	**500**	**1%**	8452,44	8868,85	7779,86	1638,65	867,08
		5%	8475,89	9903,84	7675,90	1491,57	886,19
		15%	8477,15	9901,18	7117,31	1576,24	872,05
		30%	8475,68	7526,32	6454,66	1225,47	882,30
	1000	**1%**	8246,80	9664,11	7641,75	2033,94	1060,89
		5%	8245,80	7658,50	7656,50	1783,29	1054,59
		15%	8597,56	7656,75	7282,32	2127,83	963,81
		30%	8317,66	7654,16	7156,10	2237,97	Not Available
insurance	**500**	**1%**	3162,33	3611,87	640,01	627,35	387,31
		5%	3183,78	3488,88	606,14	733,87	427,30
		15%	3161,62	Timeout	611,63	546,15	390,16
		30%	3149,41	Timeout	588,02	664,98	382,95
	1000	**1%**	3280,80	3423,15	697,99	805,50	549,20
		5%	3288,00	Timeout	669,39	1066,19	534,44
		15%	3290,77	Timeout	744,15	855,09	618,96
		30%	3253,92	Timeout	702,25	875,94	538,54

Table A.2: T-tests for AIC+CLL

T-test – AIC$_{CLL}$						
			Experiment			
Network	Examples	Missing	BBSL vs DAHVI	BBSL vs SEM$_{GHC}$(*)	BBSL vs GS(*)	BBSL vs MMHC(*)
ALARM	500	1%	Not Available	3,16E-007	3,56E-012	5,04E-012
		5%	Not Available	3,78E-006	8,81E-013	3,47E-013
		15%	1,02E-005	3,63E-009	4,59E-012	1,21E-009
		30%	3,75E-008	1,60E-008	4,17E-016	2,93E-011
	1000	1%	Not Available	1,19E-004	2,94E-008	5,33E-006
		5%	Not Available	5,50E-006	2,59E-013	3,52E-007
		15%	1,81E-011	2,16E-005	4,32E-011	1,05E-008
		30%	1,63E-007	9,32E-007	1,45E-008	1,62E-006
Child	500	1%	2,12E-007	2,26E-013	1,31E-007	2,63E-013
		5%	4,67E-007	5,65E-013	2,40E-007	2,48E-017
		15%	2,10E-006	6,32E-012	1,82E-005	5,45E-015
		30%	Timeout	2,11E-012	8,64E-010	6,66E-016
	1000	1%	3,95E-003	2,00E-012	9,88E-001	8,91E-013
		5%	2,25E-006	2,95E-011	1,63E-001	3,51E-013
		15%	6,18E-005	6,53E-015	8,78E-001	7,55E-013
		30%	Timeout	1,91E-012	4,06E-004	1,02E-014
Hailfinder	500	1%	7,23E-007	6,84E-002	3,01E-013	1,52E-017
		5%	1,14E-022	1,27E-001	3,74E-014	2,36E-020
		15%	3,00E-021	2,31E-003	3,99E-017	2,41E-020
		30%	1,10E-020	8,78E-004	1,73E-015	6,97E-021
	1000	1%	3,30E-022	1,30E-001	8,41E-012	1,61E-020
		5%	1,44E-017	2,29E-001	6,29E-013	3,93E-021
		15%	1,62E-020	6,49E-003	2,02E-012	3,95E-021
		30%	2,40E-007	6,06E-002	3,06E-011	-
insurance	500	1%	6,89E-010	7,64E-012	8,85E-017	Not Available
		5%	1,15E-004	6,15E-015	9,87E-015	8,15E-018
		15%	Timeout	1,32E-014	1,17E-015	1,38E-017
		30%	Timeout	3,59E-014	1,08E-013	1,93E-018
	1000	1%	3,67E-006	5,80E-014	4,45E-017	1,06E-020
		5%	Timeout	1,75E-015	2,96E-007	5,96E-016
		15%	Timeout	1,21E-013	4,24E-013	9,05E-021
		30%	Timeout	1,98E-014	1,52E-010	5,37E-016

Table A.3: Detailed Average AIC+CLL Increase

AIC$_{CLL}$ (Difference)							
					Experiment		
Network	Examples	Missing	BBSL	DAHVI	SEM$_{GHC}$(*)	GS(*)	MMHC(*)
ALARM	500	1%	-131,51	Not Available	-895,89	187,44	45,83
		5%	-283,08	Not Available	-926,92	-47,86	41,80
		15%	-312,96	-90,09	-1058,29	-25,38	-17,92
		30%	-217,67	-131,01	-946,55	11,36	24,24
	1000	1%	-576,45	Not Available	-968,40	-45,48	-29,82
		5%	-362,25	Not Available	-831,57	-116,26	-52,31
		15%	-347,29	-163,43	-880,96	-49,69	17,28
		30%	-527,60	-83,23	-946,89	-55,92	-196,04
Child	500	1%	-104,26	-11,70	-616,14	193,82	24,97
		5%	-127,84	-30,83	-657,32	123,66	16,67
		15%	-118,89	-89,65	-643,34	312,48	13,50
		30%	-125,70	Timeout	-646,74	172,25	11,43
	1000	1%	-139,73	-100,47	-647,23	522,40	1,54
		5%	-142,61	-105,16	-643,79	438,36	10,29
		15%	-135,57	-96,92	-640,86	524,92	-2,58
		30%	-142,70	Timeout	-655,42	245,60	-7,31
Hailfinder	500	1%	-1960,83	-1044,64	-2633,41	985,86	214,29
		5%	-1991,14	8,08	-2791,13	852,98	247,59
		15%	-1979,47	10,57	-3339,31	931,08	226,89
		30%	-1981,40	10,23	-4002,42	581,66	238,49
	1000	1%	-2423,43	-1585,19	-3028,47	1178,09	205,04
		5%	-2403,29	9,78	-2992,58	935,86	207,17
		15%	-2066,77	10,23	-3382,01	1279,19	115,18
		30%	-2330,15	10,50	-2445,91	1376,46	Not Available
insurance	500	1%	-556,06	-77,32	-3078,37	274,95	34,91
		5%	-541,69	-210,41	-3119,33	377,57	71,00
		15%	-559,22	Timeout	-3109,21	192,72	36,73
		30%	-569,41	Timeout	-3130,80	309,08	27,05
	1000	1%	-505,04	-43,68	-3087,85	272,30	16,00
		5%	-499,20	Timeout	-3117,80	527,48	-4,28
		15%	-488,59	Timeout	-3035,20	319,53	83,40
		30%	-519,02	Timeout	-3070,68	347,22	9,83

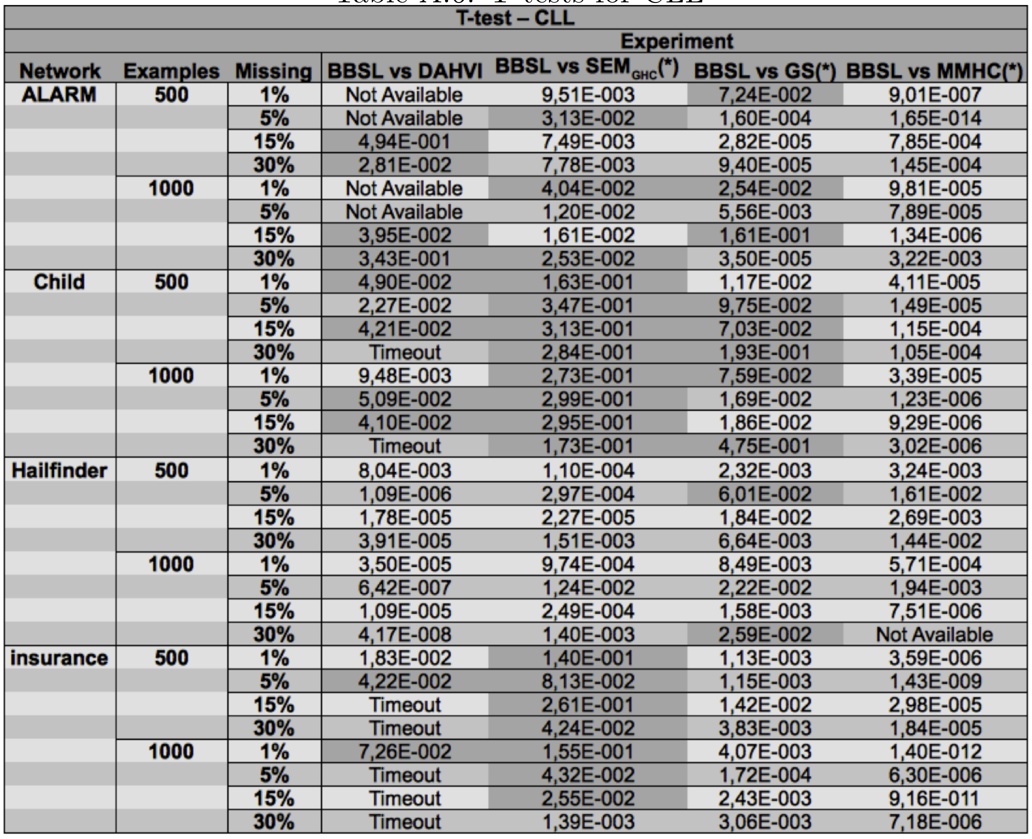
Table A.4: T-tests for AIC+CLL (Difference)

T-test – AIC$_{CLL}$ (Difference)						
			Experiment			
Network	Examples	Missing	BBSL vs DAHVI	BBSL vs SEM$_{GHC}$(*)	BBSL vs GS(*)	BBSL vs MMHC(*)
ALARM	500	1%	Not Available	3,16E-007	1,03E-007	6,60E-005
		5%	Not Available	3,78E-006	6,79E-007	7,15E-009
		15%	1,02E-005	3,63E-009	5,49E-007	9,00E-005
		30%	3,75E-008	1,60E-008	4,77E-009	1,26E-005
	1000	1%	Not Available	1,19E-004	3,31E-008	3,67E-006
		5%	Not Available	5,50E-006	3,84E-008	4,21E-004
		15%	4,31E-010	2,16E-005	6,19E-007	4,52E-006
		30%	4,22E-007	9,32E-007	2,06E-007	1,09E-003
Child	500	1%	2,12E-007	2,26E-013	5,77E-007	4,29E-008
		5%	4,67E-007	5,65E-013	1,71E-005	8,55E-010
		15%	9,62E-005	6,32E-012	3,99E-007	7,82E-011
		30%	Timeout	2,11E-012	3,67E-008	4,07E-009
	1000	1%	1,29E-003	2,00E-012	1,93E-005	7,51E-008
		5%	2,69E-005	2,95E-011	2,56E-006	5,82E-009
		15%	1,54E-003	6,53E-015	7,44E-005	1,89E-007
		30%	Timeout	1,91E-012	1,42E-005	9,85E-008
Hailfinder	500	1%	1,25E-009	6,84E-002	9,43E-010	2,42E-012
		5%	3,65E-019	1,27E-001	1,16E-010	4,24E-015
		15%	5,26E-019	2,31E-003	1,43E-013	2,51E-014
		30%	1,01E-020	8,78E-004	2,57E-011	1,13E-015
	1000	1%	1,12E-015	1,30E-001	8,65E-010	7,48E-015
		5%	5,51E-020	2,29E-001	3,15E-010	4,30E-017
		15%	4,40E-020	6,49E-003	1,30E-009	1,66E-014
		30%	6,51E-012	8,35E-001	4,25E-009	Not Available
insurance	500	1%	5,23E-010	7,64E-012	1,81E-012	1,57E-013
		5%	5,32E-005	6,15E-015	7,13E-011	1,72E-012
		15%	Timeout	1,32E-014	5,56E-011	8,39E-012
		30%	Timeout	3,59E-014	7,52E-010	9,90E-012
	1000	1%	1,91E-010	5,80E-014	1,17E-011	7,44E-013
		5%	Timeout	1,75E-015	1,37E-004	1,98E-008
		15%	Timeout	1,21E-013	1,42E-008	2,45E-012
		30%	Timeout	1,98E-014	1,23E-006	1,39E-009

Table A.5: Detailed Average CLL

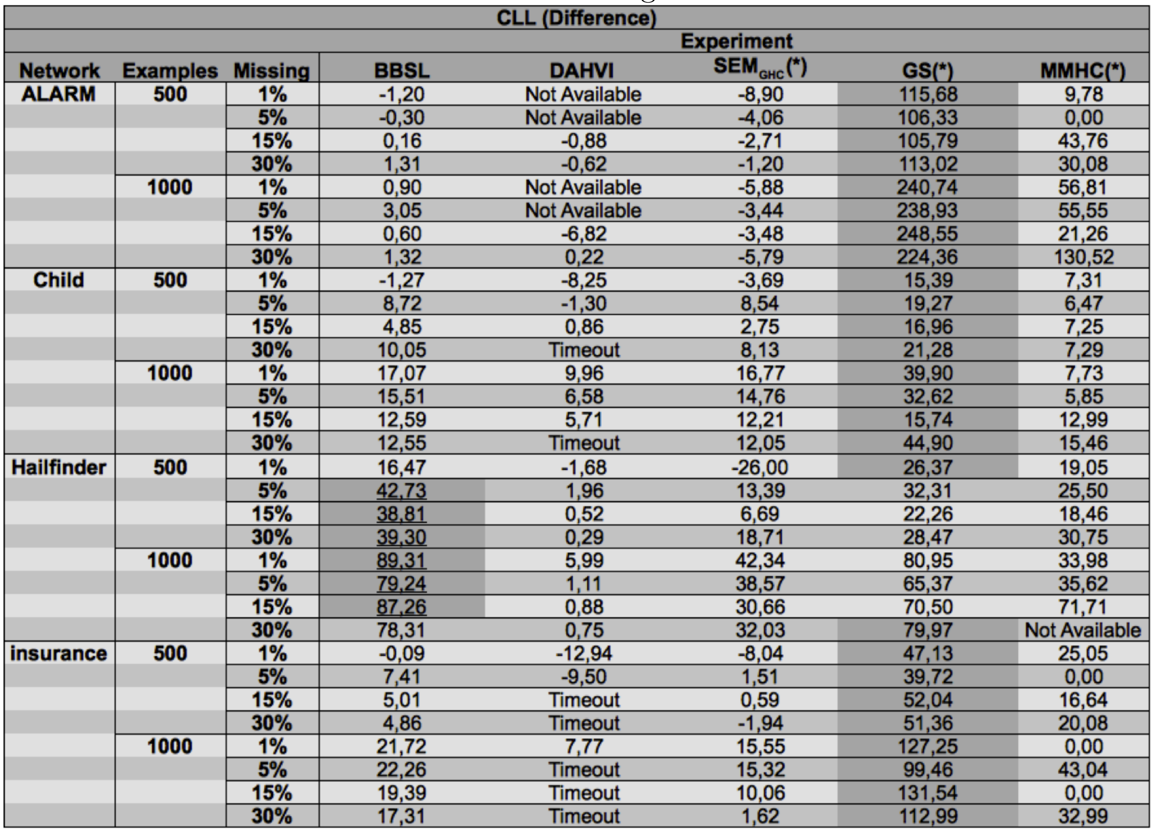
CLL							
			Experiment				
Network	Examples	Missing	BBSL	DAHVI	SEM$_{GHC}$(*)	GS(*)	MMHC(*)
ALARM	500	1%	-1,64	Not Available	-9,33	-12,34	-118,24
		5%	-0,77	Not Available	-4,53	-22,88	-129,21
		15%	-1,23	-2,27	-4,11	-22,55	-84,58
		30%	-0,38	-2,31	-2,90	-15,35	-98,29
	1000	1%	-2,61	Not Available	-9,39	-16,59	-200,52
		5%	-1,42	Not Available	-7,91	-17,69	-201,07
		15%	-1,40	-8,37	-5,49	-9,77	-237,06
		30%	-2,86	-4,00	-9,98	-26,60	-120,44
Child	500	1%	-21,51	-28,49	-23,93	-33,88	-41,95
		5%	-25,18	-35,20	-25,35	-30,12	-42,92
		15%	-22,56	-26,51	-24,66	-30,54	-40,25
		30%	-23,15	Timeout	-25,07	-26,10	-40,09
	1000	1%	-52,53	-58,69	-52,82	-60,02	-92,18
		5%	-49,41	-53,56	-50,16	-62,65	-89,41
		15%	-47,05	-51,31	-47,43	-83,35	-86,10
		30%	-51,08	Timeout	-51,58	-51,60	-81,04
Hailfinder	500	1%	-56,36	-60,09	-98,83	-77,02	-84,34
		5%	-56,98	-58,25	-86,33	-63,99	-70,79
		15%	-55,70	-57,12	-87,82	-77,32	-81,12
		30%	-55,44	-64,76	-76,03	-70,44	-68,15
	1000	1%	-112,00	-121,65	-158,98	-123,97	-170,94
		5%	-111,50	-130,25	-152,17	-135,34	-165,10
		15%	-111,10	-129,38	-167,70	-130,81	-129,61
		30%	-111,79	-128,08	-176,98	-127,78	Not Available
insurance	500	1%	-25,68	-40,29	-33,63	-40,08	-62,16
		5%	-21,73	-41,89	-27,63	-49,44	-89,15
		15%	-21,81	Timeout	-26,23	-35,68	-71,08
		30%	-20,95	Timeout	-27,75	-37,59	-68,87
	1000	1%	-37,60	-51,64	-43,77	-50,35	-177,60
		5%	-37,74	Timeout	-44,68	-80,90	-137,32
		15%	-36,69	Timeout	-46,02	-47,24	-178,78
		30%	-35,56	Timeout	-51,25	-62,37	-142,37

Table A.6: T-tests for CLL



T-test – CLL						
			Experiment			
Network	Examples	Missing	BBSL vs DAHVI	BBSL vs SEM$_{GHC}$(*)	BBSL vs GS(*)	BBSL vs MMHC(*)
ALARM	500	1%	Not Available	9,51E-003	7,24E-002	9,01E-007
		5%	Not Available	3,13E-002	1,60E-004	1,65E-014
		15%	4,94E-001	7,49E-003	2,82E-005	7,85E-004
		30%	2,81E-002	7,78E-003	9,40E-005	1,45E-004
	1000	1%	Not Available	4,04E-002	2,54E-002	9,81E-005
		5%	Not Available	1,20E-002	5,56E-003	7,89E-005
		15%	3,95E-002	1,61E-002	1,61E-001	1,34E-006
		30%	3,43E-001	2,53E-002	3,50E-005	3,22E-003
Child	500	1%	4,90E-002	1,63E-001	1,17E-002	4,11E-005
		5%	2,27E-002	3,47E-001	9,75E-002	1,49E-005
		15%	4,21E-002	3,13E-001	7,03E-002	1,15E-004
		30%	Timeout	2,84E-001	1,93E-001	1,05E-004
	1000	1%	9,48E-003	2,73E-001	7,59E-002	3,39E-005
		5%	5,09E-002	2,99E-001	1,69E-002	1,23E-006
		15%	4,10E-002	2,95E-001	1,86E-002	9,29E-006
		30%	Timeout	1,73E-001	4,75E-001	3,02E-006
Hailfinder	500	1%	8,04E-003	1,10E-004	2,32E-003	3,24E-003
		5%	1,09E-006	2,97E-004	6,01E-002	1,61E-002
		15%	1,78E-005	2,27E-005	1,84E-002	2,69E-003
		30%	3,91E-005	1,51E-003	6,64E-003	1,44E-002
	1000	1%	3,50E-005	9,74E-004	8,49E-003	5,71E-004
		5%	6,42E-007	1,24E-002	2,22E-002	1,94E-003
		15%	1,09E-005	2,49E-004	1,58E-003	7,51E-006
		30%	4,17E-008	1,40E-003	2,59E-002	Not Available
insurance	500	1%	1,83E-002	1,40E-001	1,13E-003	3,59E-006
		5%	4,22E-002	8,13E-002	1,15E-003	1,43E-009
		15%	Timeout	2,61E-001	1,42E-002	2,98E-005
		30%	Timeout	4,24E-002	3,83E-003	1,84E-005
	1000	1%	7,26E-002	1,55E-001	4,07E-003	1,40E-012
		5%	Timeout	4,32E-002	1,72E-004	6,30E-006
		15%	Timeout	2,55E-002	2,43E-003	9,16E-011
		30%	Timeout	1,39E-003	3,06E-003	7,18E-006

Table A.7: Detailed Average CLL Increase

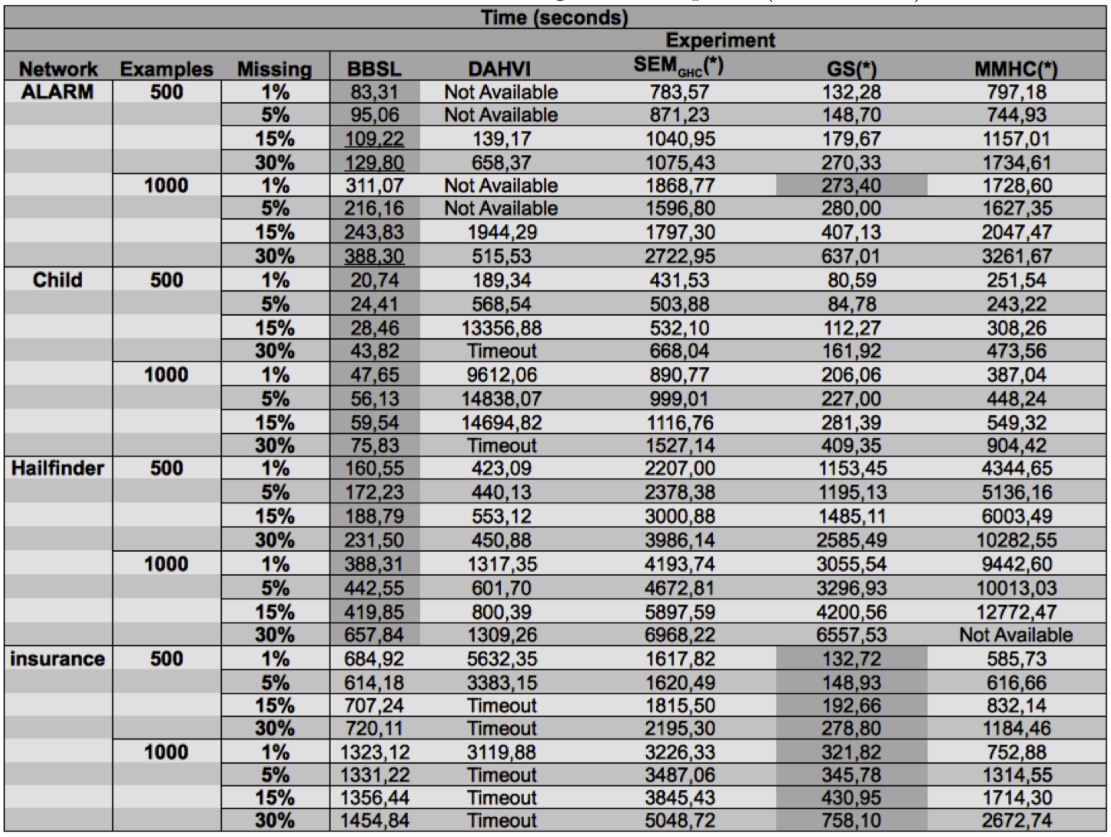


CLL (Difference)							
			Experiment				
Network	Examples	Missing	BBSL	DAHVI	SEM$_{GHC}$(*)	GS(*)	MMHC(*)
ALARM	500	1%	-1,20	Not Available	-8,90	115,68	9,78
		5%	-0,30	Not Available	-4,06	106,33	0,00
		15%	0,16	-0,88	-2,71	105,79	43,76
		30%	1,31	-0,62	-1,20	113,02	30,08
	1000	1%	0,90	Not Available	-5,88	240,74	56,81
		5%	3,05	Not Available	-3,44	238,93	55,55
		15%	0,60	-6,82	-3,48	248,55	21,26
		30%	1,32	0,22	-5,79	224,36	130,52
Child	500	1%	-1,27	-8,25	-3,69	15,39	7,31
		5%	8,72	-1,30	8,54	19,27	6,47
		15%	4,85	0,86	2,75	16,96	7,25
		30%	10,05	Timeout	8,13	21,28	7,29
	1000	1%	17,07	9,96	16,77	39,90	7,73
		5%	15,51	6,58	14,76	32,62	5,85
		15%	12,59	5,71	12,21	15,74	12,99
		30%	12,55	Timeout	12,05	44,90	15,46
Hailfinder	500	1%	16,47	-1,68	-26,00	26,37	19,05
		5%	42,73	1,96	13,39	32,31	25,50
		15%	38,81	0,52	6,69	22,26	18,46
		30%	39,30	0,29	18,71	28,47	30,75
	1000	1%	89,31	5,99	42,34	80,95	33,98
		5%	79,24	1,11	38,57	65,37	35,62
		15%	87,26	0,88	30,66	70,50	71,71
		30%	78,31	0,75	32,03	79,97	Not Available
insurance	500	1%	-0,09	-12,94	-8,04	47,13	25,05
		5%	7,41	-9,50	1,51	39,72	0,00
		15%	5,01	Timeout	0,59	52,04	16,64
		30%	4,86	Timeout	-1,94	51,36	20,08
	1000	1%	21,72	7,77	15,55	127,25	0,00
		5%	22,26	Timeout	15,32	99,46	43,04
		15%	19,39	Timeout	10,06	131,54	0,00
		30%	17,31	Timeout	1,62	112,99	32,99

Table A.8: T-tests for CLL (Difference)

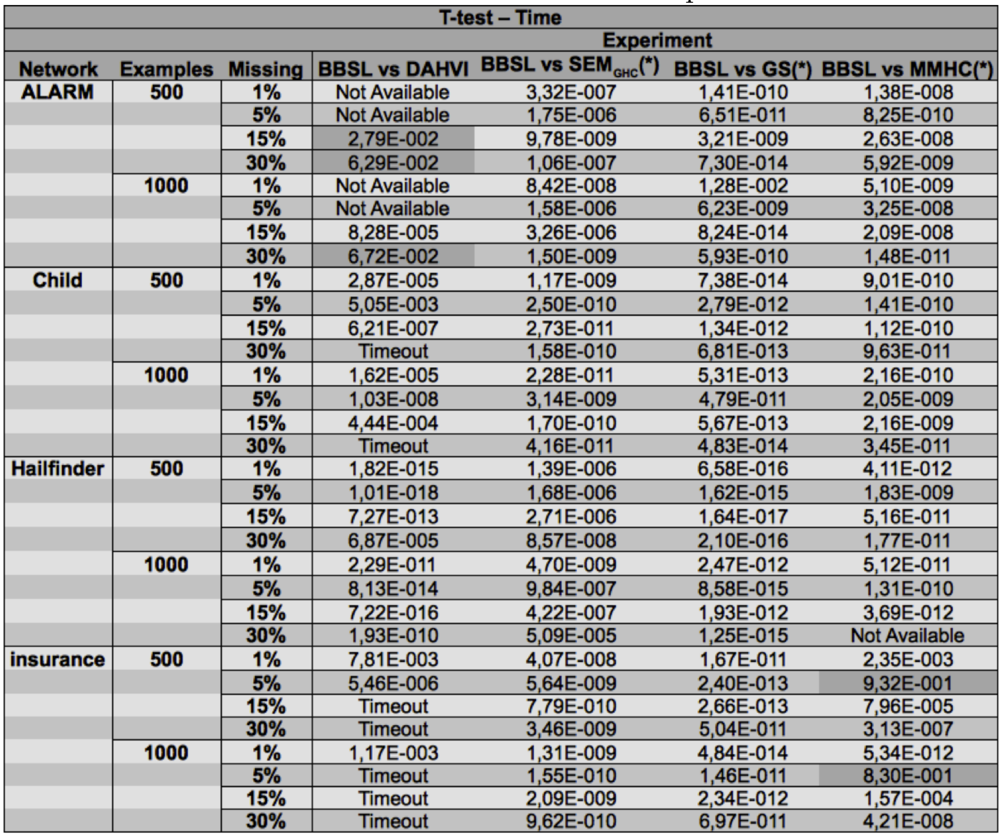
T-test – CLL (Difference)						
			Experiment			
Network	Examples	Missing	BBSL vs DAHVI	BBSL vs SEM$_{GHC}$(*)	BBSL vs GS(*)	BBSL vs MMHC(*)
ALARM	500	1%	Not Available	9,51E-003	9,01E-009	2,94E-001
		5%	Not Available	3,13E-002	5,32E-009	5,60E-001
		15%	4,94E-001	7,49E-003	1,12E-009	3,69E-002
		30%	2,81E-002	7,78E-003	1,69E-010	9,32E-002
	1000	1%	Not Available	4,04E-002	2,82E-011	8,72E-002
		5%	Not Available	1,20E-002	2,69E-011	9,28E-002
		15%	3,28E-002	1,61E-002	6,40E-011	3,51E-001
		30%	3,70E-001	2,53E-002	4,68E-013	1,73E-003
Child	500	1%	4,90E-002	1,63E-001	2,58E-003	8,97E-003
		5%	2,27E-002	3,47E-001	4,16E-002	5,11E-001
		15%	1,04E-001	3,13E-001	5,31E-002	2,56E-001
		30%	Timeout	2,84E-001	2,54E-003	3,57E-001
	1000	1%	1,16E-002	2,73E-001	2,21E-002	2,78E-002
		5%	5,12E-003	2,99E-001	3,02E-002	1,73E-002
		15%	4,17E-002	2,95E-001	8,25E-001	9,40E-001
		30%	Timeout	1,73E-001	2,79E-004	5,66E-001
Hailfinder	500	1%	5,73E-006	1,10E-004	2,42E-001	7,64E-001
		5%	1,96E-007	2,97E-004	1,64E-001	1,35E-002
		15%	1,53E-007	2,27E-005	4,57E-002	3,07E-002
		30%	6,34E-008	1,51E-003	1,25E-001	1,30E-001
	1000	1%	1,89E-009	9,74E-004	3,05E-001	5,65E-003
		5%	2,57E-009	1,24E-002	2,42E-001	5,84E-003
		15%	1,73E-010	2,49E-004	1,11E-001	9,57E-002
		30%	5,01E-009	1,00E-002	8,51E-001	Not Available
insurance	500	1%	2,02E-002	1,40E-001	5,99E-006	1,69E-003
		5%	6,45E-002	8,13E-002	1,09E-003	6,31E-002
		15%	Timeout	2,61E-001	7,35E-007	8,26E-002
		30%	Timeout	4,24E-002	3,21E-007	2,95E-002
	1000	1%	8,29E-002	1,55E-001	2,05E-008	7,14E-005
		5%	Timeout	4,32E-002	4,65E-005	2,05E-001
		15%	Timeout	2,55E-002	1,24E-008	1,04E-005
		30%	Timeout	1,39E-003	6,15E-007	1,97E-001

Table A.9: Detailed Average Time Spent (in seconds)



Time (seconds)								
			Experiment					
Network	Examples	Missing	BBSL	DAHVI	SEM$_{GHC}$(*)	GS(*)	MMHC(*)
ALARM	500	1%	83,31	Not Available	783,57	132,28	797,18
		5%	95,06	Not Available	871,23	148,70	744,93
		15%	109,22	139,17	1040,95	179,67	1157,01
		30%	129,80	658,37	1075,43	270,33	1734,61
	1000	1%	311,07	Not Available	1868,77	273,40	1728,60
		5%	216,16	Not Available	1596,80	280,00	1627,35
		15%	243,83	1944,29	1797,30	407,13	2047,47
		30%	388,30	515,53	2722,95	637,01	3261,67
Child	500	1%	20,74	189,34	431,53	80,59	251,54
		5%	24,41	568,54	503,88	84,78	243,22
		15%	28,46	13356,88	532,10	112,27	308,26
		30%	43,82	Timeout	668,04	161,92	473,56
	1000	1%	47,65	9612,06	890,77	206,06	387,04
		5%	56,13	14838,07	999,01	227,00	448,24
		15%	59,54	14694,82	1116,76	281,39	549,32
		30%	75,83	Timeout	1527,14	409,35	904,42
Hailfinder	500	1%	160,55	423,09	2207,00	1153,45	4344,65
		5%	172,23	440,13	2378,38	1195,13	5136,16
		15%	188,79	553,12	3000,88	1485,11	6003,49
		30%	231,50	450,88	3986,14	2585,49	10282,55
	1000	1%	388,31	1317,35	4193,74	3055,54	9442,60
		5%	442,55	601,70	4672,81	3296,93	10013,03
		15%	419,85	800,39	5897,59	4200,56	12772,47
		30%	657,84	1309,26	6968,22	6557,53	Not Available
insurance	500	1%	684,92	5632,35	1617,82	132,72	585,73
		5%	614,18	3383,15	1620,49	148,93	616,66
		15%	707,24	Timeout	1815,50	192,66	832,14
		30%	720,11	Timeout	2195,30	278,80	1184,46
	1000	1%	1323,12	3119,88	3226,33	321,82	752,88
		5%	1331,22	Timeout	3487,06	345,78	1314,55
		15%	1356,44	Timeout	3845,43	430,95	1714,30
		30%	1454,84	Timeout	5048,72	758,10	2672,74

Table A.10: T-tests for Time Spent



T-test – Time						
			Experiment			
Network	Examples	Missing	BBSL vs DAHVI	BBSL vs SEM$_{GHC}$(*)	BBSL vs GS(*)	BBSL vs MMHC(*)
ALARM	500	1%	Not Available	3,32E-007	1,41E-010	1,38E-008
		5%	Not Available	1,75E-006	6,51E-011	8,25E-010
		15%	2,79E-002	9,78E-009	3,21E-009	2,63E-008
		30%	6,29E-002	1,06E-007	7,30E-014	5,92E-009
	1000	1%	Not Available	8,42E-008	1,28E-002	5,10E-009
		5%	Not Available	1,58E-006	6,23E-009	3,25E-008
		15%	8,28E-005	3,26E-006	8,24E-014	2,09E-008
		30%	6,72E-002	1,50E-009	5,93E-010	1,48E-011
Child	500	1%	2,87E-005	1,17E-009	7,38E-014	9,01E-010
		5%	5,05E-003	2,50E-010	2,79E-012	1,41E-010
		15%	6,21E-007	2,73E-011	1,34E-012	1,12E-010
		30%	Timeout	1,58E-010	6,81E-013	9,63E-011
	1000	1%	1,62E-005	2,28E-011	5,31E-013	2,16E-010
		5%	1,03E-008	3,14E-009	4,79E-011	2,05E-009
		15%	4,44E-004	1,70E-010	5,67E-013	2,16E-009
		30%	Timeout	4,16E-011	4,83E-014	3,45E-011
Hailfinder	500	1%	1,82E-015	1,39E-006	6,58E-016	4,11E-012
		5%	1,01E-018	1,68E-006	1,62E-015	1,83E-009
		15%	7,27E-013	2,71E-006	1,64E-017	5,16E-011
		30%	6,87E-005	8,57E-008	2,10E-016	1,77E-011
	1000	1%	2,29E-011	4,70E-009	2,47E-012	5,12E-011
		5%	8,13E-014	9,84E-007	8,58E-015	1,31E-010
		15%	7,22E-016	4,22E-007	1,93E-012	3,69E-012
		30%	1,93E-010	5,09E-005	1,25E-015	Not Available
insurance	500	1%	7,81E-003	4,07E-008	1,67E-011	2,35E-003
		5%	5,46E-006	5,64E-009	2,40E-013	9,32E-001
		15%	Timeout	7,79E-010	2,66E-013	7,96E-005
		30%	Timeout	3,46E-009	5,04E-011	3,13E-007
	1000	1%	1,17E-003	1,31E-009	4,84E-014	5,34E-012
		5%	Timeout	1,55E-010	1,46E-011	8,30E-001
		15%	Timeout	2,09E-009	2,34E-012	1,57E-004
		30%	Timeout	9,62E-010	6,97E-011	4,21E-008