Learning Graphical Models Using Prior Knowledge

Eman Marzouq Aljohani

PHD

University of York
Computer Science

April 2015

Abstract

Graphical models represent conditional independence relationships between variables, including, for example, those between the various symptoms and causes of a disease. An important topic in the area of machine learning is learning these types of models from data. In some applications, it is crucial to include information that is not contained in the data, i.e. prior information. The aim of this research is to design an efficient algorithm that utilises prior knowledge in a manner which allows users to express what they know about the problem domain. This involves creating a system where the input is composed of prior knowledge, together with data, connected to a Bayesian learning algorithm. Our main aim is to facilitate the design of an algorithm that uses prior knowledge ahead of time, in order to both speed up the process of learning and ensure that the learning is more accurate.

List of contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I thank and praise Allah (God) the Almighty for giving me the courage and ability to seek knowledge and complete this thesis.

I would like to express my special appreciation and thanks to my Supervisor Dr James Cussens who has been a tremendous mentor to me. I would like to thank him for encouraging my research and allowing me to grow as a research scientist. His advice on my research and my career have been priceless. I would also like to thank Daniel Kudenko for providing me with comments and suggestions throughout the milestones of this research.

I would like to thank my family for all their love and encouragement. During this research, my parents Marzouq and Fozyah Aljohani have been with me through their prayers and encouragement. Words are not enough to express my gratitude to them. Finally yet importantly, I would like to thank my loving and patient husband, whose faithful support during this PhD is so appreciated, and my daughter Lujain, to whom I did not give enough whilst working on this thesis; I dedicate this thesis to you.

Declaration

This thesis has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree other than Doctor of Philosophy of the University of York. This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by explicit references. Some parts of this thesis were presented at the following events:

- Eman Aljohani and James Cussens. Informative priors for learning in graphical models. In AIGM workshop, 2014.

1 Introduction And Motivation

This chapter shows how machine learning is used to learn a well-known set of graphical models – Bayesian networks – and is concerned with learning the structure of Bayesian networks and parameter estimation. The thesis gives the necessary background for graphical models by giving simple examples of medical diagnosis problems. Section 1.2 gives a detailed account of the motivation and the research hypothesis behind the work in this thesis, which is based on a well-known graphical representation for learning Bayesian networks specifically, using prior knowledge. Finally, an overview of the thesis is given in Section 1.3.

1.1 Introduction

A Bayesian network (BN) is a graphical model that encodes the dependencies between variables, where nodes represent the random variables and edges represent the probabilistic dependence between variables (discussed in detail in section 2.2). The rest of this section will use the following simple medical diagnosis problem.

Let's take an example of a lung cancer problem: A patient has been suffering from dyspnoea and visits the doctor. The patient is concerned that he may have lung cancer. However, the doctor knows that other diseases cause dyspnoea, as well as lung cancer. Additional information includes whether or not the patient is a smoker, and if the patient has been exposed to pollution that might increase the chances of having cancer. In addition, a positive X-ray would reveal whether the patient has lung cancer (Korb & Ann E. Nicholson, 2003).

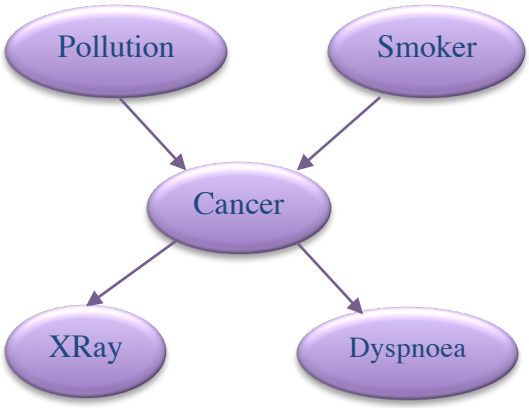

Figure 1: A Bayesian network for the lung cancer problem.

In this medical diagnosis example, two nodes should be connected directly if one affects or causes the other, with the edge specifying the direction of the effect. On other hand, having cancer will affect the patient's breathing, which increases the probability of a positive X-ray result.

A node is a parent of a child if there is an edge from the parent to the child. From a set of directed nodes, we can say that the Cancer node has two parents: Pollution and Smoker. Furthermore, node Smoker is an ancestor of both X-ray and Dyspnoea, as it appears earlier in the directed set. Whereas, node X- ray is a child of Cancer and a descendant of Smoker and Pollution, as it comes later in the directed set.
A Bayesian networks offer complete representations of probability distributions over their variables. Therefore, this indicates that they can be conditioned on any subset of variables, which supports any direction of reasoning. Bayesian networks allow us to perform diagnostic or predictive reasoning.

A doctor observes Dyspnoea and then updates his belief about Cancer. If the patient is a Smoker, this is an example of reasoning from symptoms to cause. However, if the doctor knows that the patient is a smoker, then the doctor also knows this will increase the probability of the patient having cancer. This is an example of predictive reasoning.

On the other hand, if we already know that the patient has cancer, then knowing whether they are a smoker will not make any difference to the probability of dyspnoea. Thus, dyspnoea is conditionally independent of smoking given the patient has cancer.

However, learning an optimal Bayesian networks from a given set of data is a computationally difficult problem. In structure learning algorithms, we find that for every possible edge in the network there is a question about whether to add the edge in the final network and in which direction. For larger problems, we have to resort to heuristic search approaches. In this thesis, we will use a hill climbing algorithm.

The number of possible network structures grows exponentially with every possible subset of edges, which could represent a network structure. Even restricting the structure learning so that it only has k parents (for k >1) has been proven to be NP-Complete (discussed in section 3.5). There is no efficient polynomial-time algorithm for searching the space of possible network structures (unless P=NP) in order to find a network structure that best fits with the data.

1.2 Motivation

This thesis investigates how different sorts of prior knowledge are incorporated into the developed learning algorithms. There are different types of prior knowledge, and different people have various types and levels of knowledge about a particular problem domain. However, given that a wide variety of problems need to be solved, the aim of this research is to develop a way of using the prior knowledge provided.

The main goal of this thesis is to create an algorithm that uses prior knowledge as input data, together with data, and builds a Bayesian network with a high posterior probability. Consequently, including the prior knowledge will make learning the Bayesian network much easier. Sometimes, working out what the right prior knowledge is may be difficult, but this thesis is not concerned with determining what the right prior knowledge is. Instead, the goal is to include all of the information and implicitly incorporate it in a prior distribution. This thesis discusses what sort of prior knowledge will be allowed and the Bayesian structure-learning algorithm that is used.

There are many types of prior knowledge, including the knowledge of whether or not node A is a parent of node B and known topological ordering. The most challenging type of prior knowledge, and the main subject of this research, is known ancestor relations and conditional independence. However, the main issue is that more complicated prior knowledge cannot be incorporated into a *local score* (as discussed in section 4.3). In addition to this, once we have complicated prior knowledge, just using hill climbing without changing it will fail because it will constantly generate networks that are not allowed. Therefore, we need to add some intelligence.

As learning Bayesian networks is NP-hard and these exact learning approaches will not scale to bigger datasets, exact leaning approaches are not the answer to every problem due to scalability issues. Thus, we have to use an approximate approach and a greedy approach such as hill climbing. Consequently, we need to explore improvements to hill climbing.

The Bayesian structure-learning problem is described as an optimisation problem. This thesis follows a search and score approach which builds upon a hill-climbing algorithm. It is difficult to create a good Bayesian network manually, and in many cases (applications), we have prior knowledge about a variable as well as data. Therefore, there is some motivation for designing an algorithm that takes advantage of prior knowledge. The aim of this thesis is to create an algorithm that uses prior knowledge as input data while simultaneously dealing with bigger problems.

1.3 Thesis Overview

This thesis is organised as follows:

Chapter 2 provides the necessary background. It introduces the field survey, and then provides a review of graphical models and probability theory. It also discusses the two main representations of graphical models: Bayesian networks and Markov networks. Next, it introduces the two main methods for handling the problem of parameter estimation for Bayesian networks: one is maximum likelihood estimation (MLE), and the other is based on the Bayesian approach. The directed acyclic graph represents the structure in a Bayesian network, and the values of the conditional probability distribution are the parameters. It also aims to provide an illustration of the form of prior and posterior distributions when we are in a situation where we need to express our uncertainty concerning a beta distribution (as in the case of a binary parameter) and a Dirichlet distribution (as in the case of multinomial variables).

Chapter 3 describes Bayesian network structure learning. It investigates the three main approaches used for Bayesian structure learning: constraint-based, score-based, and the Bayesian model-averaging approach. This chapter attempts to illustrate why it is important to study prior knowledge, and the various approaches that have been considered. It highlights different, related work on Bayesian structure learning approaches, and discusses the theoretical limits of learning Bayesian networks. Finally, the chapter examines some existing applications for Bayesian network learning.

Chapter 4 proposes an algorithm to learn Bayesian networks from data. It presents Cowell's (2009) approach to the exact learning of the maximum likelihood Bayesian network. It introduces the developed learning algorithm, which is a hill-climbing algorithm prior knowledge (HCPK). First, it shows how the directed acyclic graph (DAG) is represented. Then, it explains how cycle checking and the scoring function are used in the developed learning algorithm. Finally, it gives a detailed discussion of the search procedure in the HCPK.

Chapter 5 is about the main contribution of this thesis: presenting an algorithm that can incorporate different types of prior to the developed algorithm. This section introduces prior information and highlights the differences between hard and soft prior information. It then describes how different sorts of prior knowledge are incorporated into the developed learning algorithms.

Chapter 6 reviews the research thesis objectives and results. It emphasises the contributions that this research has made, and presents experiments conducted using the developed algorithm on Dynamic Programming and HCPK, with and without prior knowledge.

Chapter 7 presents a summary of all the chapters included in the thesis. It identifies the limitations of the current work, and provides some possible directions for future study.

2 Background

This chapter covers the background about the fields of machine learning, graphical models, and probability theory. Section 2.1 provides an introduction to machine learning, and introduces probability theory and how it is linked to *graphical models*. An overview of the different types of graphical models that are most commonly used as representations of variable relationships is presented. The main types of graphs are a Bayesian network (discussed in section 2.2) and a Markov network (discussed in section 2.3). Section 2.4 explains the two main methods used to handle the problem of parameter estimation for Bayesian networks.

2.1 Introduction

A machine typically learns whenever it changes its structure, data, or software. These changes are based on inputs or external information. However, learning similar to intelligence covers a wide range of processes that are difficult to define precisely. Machine learning denotes the changes in any system that performs some tasks linked with artificial intelligence. Examples of such tasks are recognition, diagnosis, and prediction, and the changes might be either improvements to the system's performance or the installation of new systems. In addition to this, different methods of learning are possible. One of the reasons machine learning is important is because "The amount of knowledge available about certain tasks might be too large for explicit encoding by humans. Machines that learn this knowledge gradually might be able to capture more of it than humans would want to write down" (Nilsson, 2005).

In general, probability theory is the mathematical study of uncertainty. It plays a main part in machine learning because the design of learning algorithms typically relies on the assumption of probabilistic data. According to Murphy (2001), when probability theory and graph theory are combined, this is called *graphical models*. He goes on

to, "They provide a natural tool for dealing with two problems that occur throughout applied mathematics and engineering – uncertainty and complexity – and, in particular, they are playing an increasingly important role in the design and analysis of machine learning algorithms" (Murphy, 2001).

Generally, a probabilistic graphical model is a graph where random variables are represented by nodes, and the lack of edges denotes conditional independence assumptions. There are two main classes of graphical models: Bayesian networks (BN) and Markov networks (MN). Bayesian networks are directed graphical models that are common in artificial intelligence (AI) and machine learning. By contrast, Markov networks are undirected graphical models that mainly concern the physics and vision communities. A *chain graph* has both undirected and directed links.

The basic concept of probability theory can be illustrated by considering the idea of finding the probability that a cancer patient will react to some particular chemotherapy, or by observing the outcome of rolling a pair of dice. These possible outcomes are called *sample points*. The set of all possible sample points in a situation of interest is called a *sample space*. The sample points in a sample space must be mutually exclusive and collectively exhaustive. A *probability measure,* $p(\cdot)$, is a function on *subsets* of a sample space Ω, which are called events. The values of $p(\mathrm{A})$, $p(\mathrm{A} \cup \mathrm{B})$, and $p(\Omega)$ indicate the probabilities of the respective events (for $\mathrm{A}, \mathrm{B} \subseteq \Omega$). The function $p(\cdot)$ is a *measure* with the following properties:

1. **"Definition**: A *probability measure* on a sample space Ω is a function mapping subsets of Ω to the interval [0, 1] such that:" (Krause, 1998)
2. $A \subseteq \Omega, \mathrm{P}(A) \geq 0, A$ any event
3. $P(\Omega) = 1$
4. For any countably infinite collection of disjoint subsets of $\Omega, A_k, \mathrm{K} = 1, .., \infty$

$P(\bigcup_{k=1}^{\infty})\ A_K = \sum_{k=1}^{\infty} P(A_k)$

Furthermore, according to Krause (1998), probability theory offers a method through which the probabilities of events are updated as we obtain evidence. In probability theory, conditional probability and Bayes' theorem are very important. Bayes' theorem has an enormous use in practical fields; for example, in medical diagnosis, to find the probability of a disease, given a symptom. Bayes' theorem is also used to manage some situations where an event is the parameter value or particular structure for a given data set. The probability of a particular event A that is conditional on event B is expressed as $\mathrm{P(A|B)}$. Bayes' theorem is:

$$\mathrm{P(A|B)} = \frac{P(B|A)P(A)}{P(B)}$$

Where $P(A)$ is the prior probability, $P(B|A)$ is the likelihood of observing B (which is conditional on A), and $P(B)$ is the probability of the evidence B.

In probability theory, a set of events is often defined using *random variables* (Russell & Norvig, 2010). Every value of a random variable takes some domain. These random variables can be Boolean, discrete, or continuous. In the case of a Boolean variable, it generally has the value $true\ or\ false$. For example, when throwing dice the instance when a double is rolled can be written as $doubles = True$. Discrete variables have a countable domain; for example, $weather$ has the domain <sunny, snow, rainy, cloudy>; $weather =$ sunny. Continuous variables take on values from real numbers. We can express the probability that a random variable lies within some particular range of value x, for example, today's temperature $x \geq 9$.

In addition, probability theory is a suitable basis for uncertain reasoning. Uncertainty occurs due to computational limitations and lack of knowledge. Probabilities encode user uncertainty and are used to obtain the right decision within a domain of interest, summarising beliefs relative to the evidence. Let us consider an example of uncertain reasoning in the medical domain: diagnosing a patient's toothache. For a particular patient that has toothache, we might consider that the toothache is caused by a cavity, but this is not necessarily true because some patients may have gum problems or another of the many

problems that result in toothache. We can say that we believe there is an 80% chance (probability of 0.8) that someone who has a toothache will have a cavity. In other words, the probability that the patient has a cavity, given that he has a toothache, is 0.8. However, knowing the patient's history of other toothache problems will alter this statement, so we could say that the probability that this patient has a cavity, given that he has a toothache and other toothache problems, is 0.4.

Prior probabilities typically represent person degrees of belief $P(\text{cavity}) = 0.2$, and then later we obtain information, which is called *evidence*. Moreover, prior probability distribution of an uncertain amount is probability distribution that one can express their belief before some evidence is considered. If a person is going to the dentist because he has toothache, then $P(\text{cavity}) = 0.2$ will be updated based on observed information relating to the toothache, this is called the posterior probability $P(\text{cavity} \mid \text{toothache}) = 0.6$. Probabilistic inference, which is used for answering any queries, is the process of computing any query conditioned on observed evidence.

However, Empirical Bayes methods are used to estimate the statistical inference from the data of the prior distribution. This method is an alternative for the Bayesian method in which the prior distribution is to be fixed before observation. Although Empirical Bayes is considered different yet it resembles the complete Bayesian hierarchical model treatment. In this the higher hierarchy parameters are paired with their most possible values without being integrated. Maximum Marginal Likelihood is another name of the Empirical Bayes method and it symbolizes the hyper parameters approach.

2.2 Bayesian network

A Bayesian network (BN) is a graphical model that encodes the dependencies between variables. Furthermore, "Bayesian networks are one of the most important, efficient, and elegant frameworks for representing and reasoning with probabilistic models. They have been applied to many real-world problems in diagnosis, forecasting, automated vision, sensor fusion, and manufacturing control"(Getoor & Taskar, 2007).

As shown in Figure [2], the main representation in Bayesian networks is the directed acyclic graph (DAG): where nodes represent the random variables and edges represent the probabilistic dependence between variables (influences).

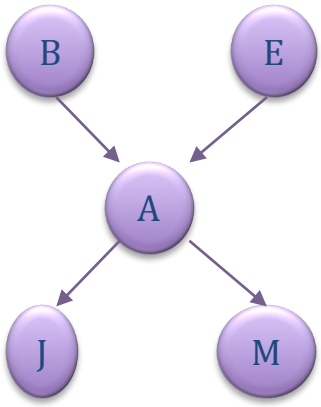

Figure 2: An example of Bayesian network

Figure [2] demonstrates an example of a BN with five random variables. B and E are represented as parents of child A, and child A is a parent of J and M. Also, the set $\{B, E\}$ of the parents of a child is identified as the parent set of that child, which is denoted by $A \leftarrow \{B, E\}$.

However, the main semantics of Bayesian networks are illustrated by the full joint distribution $P(x_1, \dots, x_n)$; the joint probability distributions for variable x: $P(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i | \text{parents}(x_i))$. Moreover, at each node there is a conditional probability distribution (CPD) for the corresponding variable given its parents $P(x_i | \text{parents}(x_i))$, which encodes the strength of dependencies.

This conditional probability distribution $P(x_i | \text{parents}(x_i))$ for discrete variables is typically expressed as a table that has an entry for each joint assignment for the corresponding variable x_i and its $\text{parents}(x_i)$. Therefore, for each node there is a conditional probability table that measures the effect of the parents on the node. The parameters are expressed as the probabilities in these conditional probability tables (CPTs), as shown in Figure [3].

For variables that have no parents, the conditional probability table conditioned on the empty set of variables, prior probabilities, and CPD becomes a marginal distribution; for example, from Figure [3], where $P(B)$ and $P(E)$ are the prior probabilities.

Meanwhile, each entry in a joint distribution is represented by the product of the CPTs in a Bayesian network (Russell & Norvig, 2010).

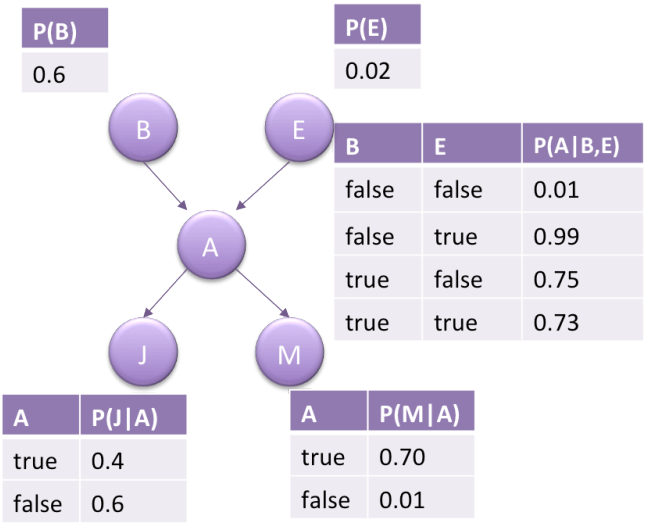

Figure 3: Conditional probability tables (CPTs).

As the DAG is the main graphical representation for a Bayesian network, there are no cycles in the graphs, which means no directed path will start and end at the same node (Koller & Friedman, 2009). This is mainly to ensure that $P(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i | \text{parents}(x_i))$ holds.

Dependencies and Independencies in Bayesian network

The dependencies and independencies in Bayesian networks are the main properties of the distribution they define, and it is important to gain an understanding about its behaviour. D-separation is one of the ways of checking for conditional independence relations, and it discovers nodes reachable from a node A, given a set of nodes Z, via active trails. The local independencies in Bayesian networks are that each node is independent of its non-descendants, given its parents. Whilst global independencies are derived from d-separation, which helps to ensure that specific sets of independencies $(A \perp B|C)$ hold in a distribution, so that a variable A is conditionally independent on a particular variable B, given variable C.

Jensen and Nielsen (2007) demonstrate that in *d-separation* there are three main patterns that illustrate whether two variables are independent in the presence of evidence. The first is known as a serial connection, and is shown in Figure [4].

Figure 4: A serial connection.

When B is not observed, A has an influence on C through B. Also, evidence about C will influence the certainty of A through B. A and C are mutually dependent. On the other hand, when B is observed, A will not provide additional information about C, so the path will be blocked. Then, A and C are independent and these are d-separated, given B. When the variable is observed, it is instantiated, which blocks communication between A and C.

The second pattern is known as a diverging connection, and is demonstrated in Figure [5].

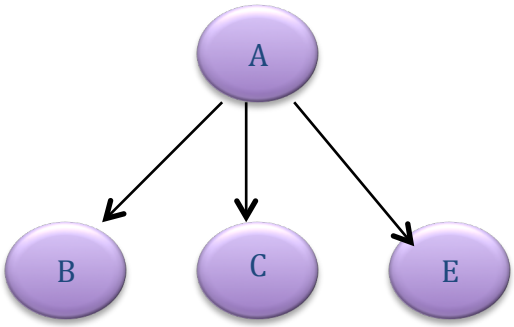

Figure 5: A diverging connection.

When A is not observed in this situation, influence is passed through to all of the children of A**.** For example, B gives us information about A, and helps to predict E and C. In this case, B, C and E are dependent, and we say that, C and E are d-connected. However, when A is observed, influence is not passed between all of the children of A because if we know A then knowing about B will not tell us anything new about C or E. In this case, B, C and E are independent, so we say that B, C and E are d-separated, given A.

Finally, the case of converging connections, as demonstrated in Figure [6].

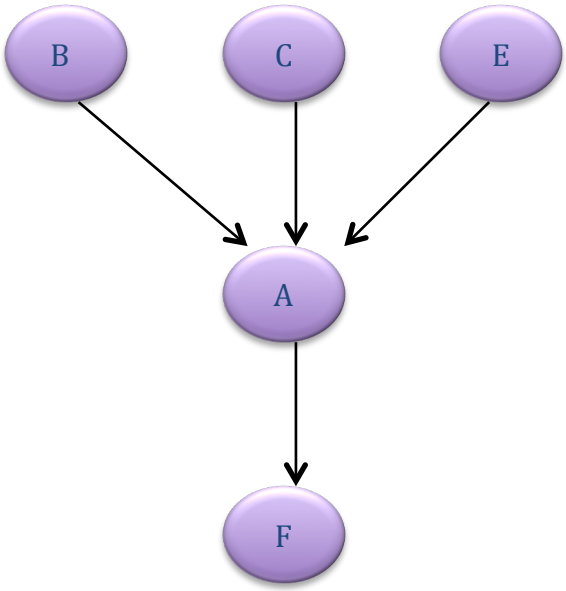

Figure 6: Converging connections.

When A is not observed, influence is blocked between all of the parents of A. The parents B, C and E are independent, and we say that B, C and E are d-separated. However, when A is observed the influence moves from B through A to affect what we believe about C, and E; so the communication is active between its parents.

Furthermore, here is an example of the converging connection in Earthquake Pearl (1988).

Figure 7: Converging connections for Earthquake Pearl (1988).

Suppose we heard an $Alarm$, and there are two possible causes: $Earthquake$ or $Burglary$, as it shown in Figure [7]. Then, someone said there was a $Burglary$; this reduces the probability of $Earthquake$. From the other perspective, if the $Alarm$ has sounded and there is a burglary, this reduces the probability of an $Earthquake$ since it has been "explained away" by the $Burglary$.

Markov equivalent

Bayesian network structures (DAGs) are equivalent if they have the same conditional independence relations. Let G_1 *and* G_2 be graphs that have the same set of nodes, such that $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$. Then the two graphs are called Markov equivalent if $A, B, C \subseteq V$ in G_1, and the nodes A and B are d-separated, given C, if, and only if, the nodes in G_2 A and B are d-separated, given C:

$$I_{G_1}(A, B|C) \Leftrightarrow I_{G_2}(A, B|C)$$

Markov equivalence can be identified using the following theorem: "two DAGs G_1 and G_2 Markov equivalent if and only if they have the same links (edges without regard for direction) and the same set of uncoupled head-to-head meeting" (Richard E. Neapolitan, 2004).

The scoring-equivalent is used with a combination of heuristic search algorithms to obtain a model. However, it is more crucial to search between equivalence classes than to search through every single network structure that is used by some approaches. In order to define the search space, the general representation needs to be stated. There is a *skeleton* for any directed acyclic graph, which means that for each edge it disregards its directionality. Another representation is a directed acyclic graph G that holds the direct edges $x \rightarrow y$ and $z \rightarrow y$ (as shown in figure [8]) in such a way that every triple sequence of nodes (x, y, z), and x and z are not adjacent in G, is called a *v-structure* (D. M. Chickering, 2002).

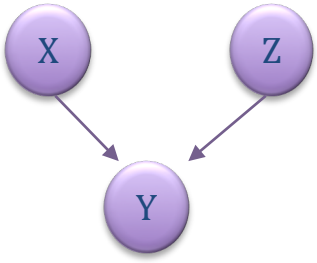

Figure 8: V-Structure.

However, if two directed acyclic graphs have exactly equal *skeletons* and exactly equal *v-structures,* they are said to be *equivalent*. Uncoupled head-to-head meeting is also called immorality. (Koller & Friedman, 2009) define an immorality as follows: "A v-structure $x \rightarrow z \leftarrow y$ is an immorality if there is no direct edge between x and y. If there is such an edge, it is called a covering edge for the v-structure". Moreover, two DAGs are Markov equivalent if they encode the same conditional independence relations, as illustrated in Figure [9].

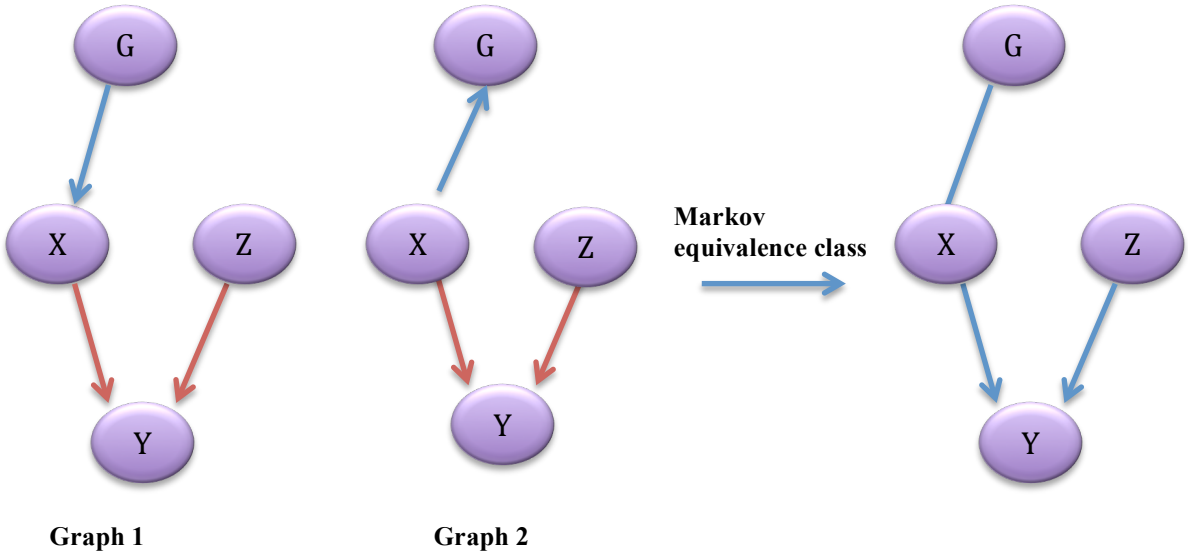

Figure 9: DAG pattern for a Markov equivalence class.

(Gillispie & Perlman, 2001) shows the efficiency of searching with Markov equivalence classes rather than searching with DAGs. They discuss an algorithm that enumerates the equivalence classes of DAGs and records their sizes. The software generates DAGs and then places them into the relevant equivalence classes. As a result, the software shows that the effective speed was significantly improved.

2.3 Markov Network

Markov networks (MN) are another fundamental class of graphical model representation, built on the basis of undirected graphs, as shown in Figure [10] (Koller & Friedman, 2009). Nodes in a Markov network represent random variables, and edges represent interaction among the neighbouring variables. Undirected graphs can also be used to represent dependency interactions, and are useful in modelling domains where the neighbouring variables seem symmetrical. Also, Markov networks are useful when one cannot naturally assign directionality to the interaction among the neighbouring variables, as you need to assign a directionality to each influence for a Bayesian network.

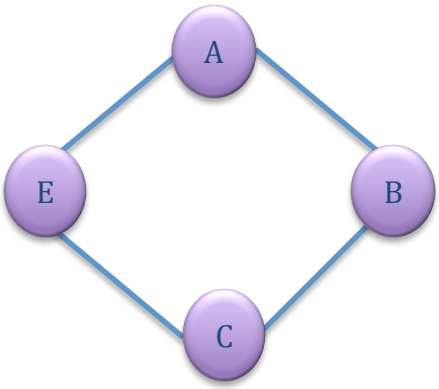

Figure 10: Markov network.

A clique is defined as follows: "A subgraph over X is complete if every two nodes in X are connected by some edges. The set X is often called a clique; we say that a clique X is maximal if for any superset of nodes $Y \supset X$, Y is not a clique" (Koller & Friedman, 2009).

However, the joint probability is in a product form in both Markov and Bayesian networks, where the CPTs are in a Bayesian network. As such, the probability distribution is denoted as a product of clique potentials.

It is necessary to associate the network structure with parameters in order to obtain a distribution. Therefore, a Markov network uses Φ to obtain a distribution. A clique is a subset of nodes in Markov networks; let $(C_1), \dots, (C_k)$ be the cliques in a Markov network. As shown in Figure [10], there are four cliques, which are {A, B}, {B, C},

{C, D} and {A, D}. Let C be a clique. The parameterisation in the Markov network is performed by a set of factors,Φ, and each factor is assigned to a clique C_k. These factors are called clique potentials $\Phi_1(C_1), \dots, \Phi_k(C_k)$. A clique potential Φ_c maps each joint instantiation of the variables in C to non-negative real numbers.

$$let\ \Phi = \{\Phi_1(C_1), \dots, \Phi_k(C_k)\}$$

A distribution $P_\Phi$is parameterised by a set of factors Φ

$$P_\Phi(X_1, \dots, X_n) = \frac{1}{Z} P'_\Phi(X_1, \dots, X_n)\ , where$$

$$P'_\Phi(X_1, \dots, X_n) = \Phi_1(C_1) \times \dots \Phi_k(C_k)$$

Where Z is the normalizing constant.

There are two types of Markov property, which are defined as local Markov properties and global Markov properties. The main idea is that nodes X are independent from nodes Y, conditioned on intervening nodes Z. Let H be an undirected graph, and let χ be the set of all nodes of H. Then for each node $X \in \chi$, the Markov blanket of X ($MB_H(X)$) is the set of neighbours of X in the graph. The local independence in a Markov network is defined as:

$$I(H) = \{(X \perp \chi - \{X\} - MB_H(X) | MB_H(X) : X \in \chi\}$$

This means that a node is conditionally independent from the remaining nodes in the network structure. On the other hand, the global independencies assumption is that if there is no active path between any $X \in \boldsymbol{X}$ and $Y \in \boldsymbol{Y}$**,** given a set of nodes $\boldsymbol{Z}$, then we can say that X is *separated* Y in Markov network H, which is expressed by $sep_H(\boldsymbol{X}; \boldsymbol{Y} | \boldsymbol{Z})$. Moreover, "Let H be a Markov network structure, and Let X_1 -…-X_k be a path in H. let $\boldsymbol{Z} \subseteq X$ be a set of observed variables. The path X_1 -…-$X_k$is active given $\boldsymbol{Z}$ if none of the $X_i's, i = 1,,,k,$ is in Z " (Koller & Friedman, 2009)**.** Also, a path is defined as if we have a path X_1 -…-X_H in a graph H and for every $i = 1, \dots k-1$, we have that either $X_i \rightarrow X_{i+1}\ or\ X_i - X_{i+1}$. The global independencies are denoted as:

$$I(H) = \{\boldsymbol{X} \perp \boldsymbol{Y} | \boldsymbol{Z}\} : sep_H(\boldsymbol{X}; \boldsymbol{Y} | \boldsymbol{Z})\}.$$

There are conditional independence relations that can be expressed with a Markov network but cannot be expressed with a Bayesian network. For example, in the Markov network shown in Figure [10], the only independence relations are $(A \perp D|B, C)$ and $(B \perp C|A, D)$; however, there is no equivalent Bayesian network to that Markov network. Moreover, the arrows in a Bayesian network do not necessarily indicate causation. The main reason for using a Markov network rather than a Bayesian network is that some conditional independence relations can be expressed using a Markov network but not a Bayesian network. .

2.4 Bayesian Estimation of Probabilities

Figure 11: Thumbtack position.

Let us look at the issue of parameter estimation for Bayesian networks. First, we make an assumption that we have a fully observed data set and fixed network structure(Koller & Friedman, 2009).

Here, we will discuss the problem of thumbtack tosses as parameter learning for one variable. Flipping the thumbtack many times will result in a dataset that has heads or tails as the outcome, as shown in Figure [11]. Using this dataset, we want to estimate the probability that the next toss will land on heads or tails. Moreover, we have made a hypothesis for thumbtack tosses as ruled by some unknown parameter θ, which is used in the thumbtack tosses to illustrate the frequency of heads. A parameter is a real number. In general, a set of parameters for a probability model specifies a particular probability distribution; for example, given a Bayesian network model defined by a DAG, the parameters are the CPTs. From the thumbtack toss example in Figure [11], the parameter is the probability of the thumbtack landing as heads. It is not necessarily connected to any Bayesian network.

We have also created another hypothesis: that the data instances in thumbtack tosses are independent and identically distributed (IID). As we toss the thumbtack many times, we produce a dataset that consists of heads or tails outcomes. Based on this dataset, we want to estimate the probability that the next toss will land on heads or tails (P). We also need to define the parameter space θ for the thumbtack problem within the interval [0,1], which is the probability (P) of the thumbtack (x: θ)

$$\text{thumbtack(x: } \theta\text{)} = \begin{cases} \theta & \text{x} = \text{head} \\ 1-\theta & x = tail \end{cases}$$

We have already made the assumption that the thumbtack tosses are controlled by some parameter θ. This parameter governs the frequency of heads in the thumbtack tosses. We have also made another assumption: that the data is independent and identically distributed (IID).

There are two main methods to handle the problem of parameter estimation for Bayesian networks in the frameworks of structured CPDs: one is maximum likelihood estimation and other is the Bayesian approach.

Using a Bayesian network to represent a distribution, we need to link the network structure with a set of parameters. The DAG represents the structure in a Bayesian network and the values of the CPD are the parameters. We turn now to provide an illustration of the form of prior and posterior distributions. If we are in a situation where we need to express our uncertainty, one convenient choice is to use a beta distribution (in the case of binary parameters), and Dirichlet distributions (in the case of multinomial variables).

2.4.1 Beta distribution

Let us give an example of the forms of prior and posterior. In the case of flipping a coin, it is convenient to describe our uncertainty about the parameters using a beta distribution for binary random variables. The beta distribution is parameterised by two hyperparameters that help to control the distribution over parameter θ (Bishop, 2007). The two hyperparameters are h (related to the number of observations of heads) and t (related to the number of observations of tails). The data set D contains the number of heads h and tails t. The likelihood function here is the binomial distribution function, which is the distribution of the number m of observation of heads $x = 1$, ($m = x_1 + \cdots + x_N$) and is conditional on the size of the dataset N.

$$Bin(m|N,\theta) = \binom{N}{m} \theta^m \, (1-\theta)^{N-m}$$

Picking some initial values for h and t states the prior belief.

$$P(\theta) = Beta(\theta|h,t) = \frac{\Gamma(h+t)}{\Gamma(h)\Gamma(t)} \, \theta^{h-1} \, (1-\theta)^{t-1}$$

Where $\Gamma(x)$ is the gamma function and $\frac{\Gamma(h+t)}{\Gamma(h)\Gamma(t)}$ is used to make sure the area under the curve equals one.

The posterior distribution is also a beta distribution, and represented by multiplying the prior belief with the likelihood function. To compute the posterior distribution we make an increment of h for each *heads* outcome, and t for each tails outcome.

$$p(\theta|m,l,h,t) \propto \theta^{m+h-1}(1-\theta)^{l+t-1} \ , where\ l = N - m$$

Based on the number of observations N the probability of heads on the next toss can be determined, given the observed dataset D:

$$P(heads|D) = \frac{m+h}{m+h+l+t}$$

From the posterior distribution, we can compute the mean and variance of the beta distribution easily.

$$\mathbb{E}[\theta] = \frac{h}{h+t}$$

$$Var[\theta] = \frac{ht}{(h+t)^2(h+t+1)}$$

2.4.2 Dirichlet distribution

In contrast, what if the variable is not Boolean such that it takes *n* values (*n >2)?*. Dirichlet distributions are a generalisation of beta distributions for parameters θ_k of the multi-valued case: a multinomial distribution (Bishop, 2007). The distribution here is parameterised by a set of hyperparameters $\alpha_1, \dots, \alpha_k$.

A Dirichlet distribution allows us to illustrate our uncertainty about the value of parameters of the multi-valued case $\theta_1, \dots, \theta_k$.

The prior Dirichlet distribution of parameters conditioned on the parameters α takes this form:

$Dir(\theta|\alpha) \propto \prod_{k=1}^{k} \theta_k^{\alpha_k - 1}$ Where α is ($\alpha_1, \dots, \alpha_k$)

The likelihood function is the multinomial distribution that is the probability of parameters given the total size of the dataset N.

$$Mult(m_1, \dots, m_k|\theta, N) = \binom{N}{m_1, \dots, m_k} \prod_{k=1}^{k} \theta_k^{m_k}$$

The posterior distribution is defined by multiplying the likelihood function by the prior distribution.

$$P(\theta|D, \alpha) = Dir(\theta|\alpha + \boldsymbol{m}) \propto \prod_{k=1}^{k} \theta_k^{m_k+\alpha_k-1}$$

$$where\ \boldsymbol{m} = (m_1, \dots, m_k)^{\boldsymbol{T}}$$

This equation makes it clear that it very easy to compute the posterior distribution, as the parameters of the posterior are the parameter prior plus counting data.

2.4.3 Maximum likelihood estimation (MLE)

In the maximum likelihood estimation approach, we use the likelihood function to determine the quality for various parameter values. The maximum likelihood estimator (MLE) attempts to find the Θ that maximises the likelihood of parameter values θ relative to the datasets D. Let's take an observed dataset D of m outcomes and use it to instantiate the values $x[1], \dots, x[m]$. The likelihood function is:

$$L(\theta: D) = \prod_{m} P(x[m]): \theta)$$

Next, we choose the parameter value that maximises the likelihood:

$\widehat{\Theta} = \max_{\theta} L(\theta: D)$

However, one of the drawbacks of the MLE method is, for example, if we run an experiment of the thumbtack problem and obtain 3 heads out of 10 (the chances of seeing 3/10 heads). In this example we want to determine θ, the probability of obtaining a result of heads of this coin.

$$\frac{dL(\theta)}{d\theta} = (1-\theta)^7\ 3\theta^2 - \theta^3\ \ 7(1-\theta)^6$$

$$= \theta^2(1-\theta)^6[\ 3(1-\theta) - 7\theta]$$

$$\text{if } \frac{dL(\theta)}{d\theta} = 0, \text{ then}$$

$$3 - 10\theta = 0$$

$$\theta = \frac{3}{10} = 0.3$$

$$L(\theta) \propto \theta^3\,(1-\theta)^7$$

In other words, 0.3 leads to the highest probability of observing 3 heads out of 10 tosses. However, if we conduct another experiment where we obtain 300 heads out of 10,000 then, there will be a difference between the experiments because we have more confidence in the last experiment.

2.4.4 Bayesian Estimation

The Bayesian approach applies a prior distribution over the parameters. In addition, we express our uncertainty about the value of a parameter by placing a prior distribution over possible values of θ. Using Bayes' theorem to calculate the posterior distribution given the observed data:

$$\mathrm{P}(\theta|\mathrm{D}) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Where $P(\theta)$ is the prior distribution, $P(D)$ is the marginal likelihood/ normalising constant, and $P(D|\theta)$ is the likelihood function.

Furthermore, (Heckerman, 1996) shows that in the Bayesian approach we can express our uncertainty about the structure G by stating the discrete variables that are likely to

make it the optimal structure. The probability of estimating the next case from observed data D is:

$$P(x_{N+1}|D) = \sum_G (G|D) \int P(x_{N+1}|\theta, G)P(\theta|D, G)\, d\theta$$

The computation $P(\theta|D, G)$ is done by making the assumptions that the data are fully observed, and that we have parameter independence:

$P(\theta|D, G) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} p(\theta_{ij}\,|D, G)$

Here θ_{ij} is the multinomial parameter and q_i is the number of configurations of the variable x corresponding to the parents. We also make the assumption that the parameters vector θ_{ij} is independent and is a Dirichlet prior distribution. From Bayes' theorem:

$$P(G|D) = \frac{P(G)\, P(D|G)}{P(D)}$$

$P(D)$ is a normalising constant that does not help to make a distinction between structures, so it can be omitted. The marginal $P(D|G)$ of data conditioned on the structure can be written as:

$$P(D|G) = \int P(D|\theta, G)P(\theta|G)\ \mathrm{d}\,\theta$$

Here $P(D|\theta, G)$ is the likelihood of data given network structure and parameters. $P(\theta|G)$ is the parameter priors given network structure. The structure prior $P(G)$ defines a probability over each network structures. One of the simplest methods for placing a prior on a structure is to make the assumption that every structure has the same probability. The drawback in this assumption is that it can be incorrect and is usually used for the ease of the choice. A more complex prior is for the user to rule out some structures, and then execute the rest of the structures as a uniform prior.

Works for CPT

As shown in the section 2.2, for each node in a Bayesian network we consider each entry in its CPT. Each node X_i has a conditional probability distribution P(X_i | Parents(X_i)). For each entry in the CPT, there is a prior Dirichlet or Beta distribution over its values. This distribution is updated based on the relevant data points, which are those that were approved on the conditional probability for the parents that correspond with this CPT entry. Also, the score that used in the developed algorithm is based on Dirichlet distribution.

In this chapter, the necessary background of graphical models and probability theory was presented. In addition, the two main representations of graphical models: Bayesian networks and Markov networks were discussed. Then, the two main methods for handling the problem of parameter estimation for Bayesian networks, the maximum likelihood estimation (MLE) and the Bayesian approach, were discussed. However, what if we do not know the structure? The following chapter will discuss the problems of learning both parameters and structures that are based on fully observed data.

3 Approaches To Bayesian Network Structure Learning

This chapter covers the problems of learning both parameters and structures that are based on fully observed data. Section 3.1 discusses the three main approaches for Bayesian structure learning: constraint-based, score-based, and the Bayesian model averaging approach. Section 3.2 introduces prior knowledge and explains why it is important to study this area. Section 3.3 presents a discussion about related work on Bayesian structure learning approaches, along with different approaches that have considered some form of prior knowledge. Existing applications for Bayesian network learning are discussed in section 3.4, and the theoretical limits of learning Bayesian networks are examined in section 3.5.

3.1 Bayesian Structure Learning

The previous chapter explains the problem concerning learning the parameters of Bayesian networks based on fully observed data. An assumption was made that the network structure was fixed. But what if we do not know the structure? In this section, we will consider the problems of learning both parameters and structures that are also based on fully observed data. This section discusses the three main approaches for Bayesian structure learning; constraint-based, score-based, and the Bayesian model averaging approach.

3.1.1. Constraint-Based Approach

Given the set of conditional independencies in a probability distribution, the constraint-based approach attempts to find a DAG for which the Markov condition requires all those conditional independencies. From the dataset, we assume that the conditional independencies IND_p can be estimated in a probability distribution P. The aim of this approach is to find a DAG whose d-separations are the same as IND_p. The conditional independencies represented by d-separation are faithful to the network structure if all the conditional independencies hold the d-separation in the structure, so that all the network structures are Markov equivalent to the network structure (Richard E. Neapolitan, 2004).

In the constraint-based approach, d-separation is given to the learning algorithm and the main goal is to learn a Bayesian network that satisfies these constraints. The PC algorithm is an example of using the constraint-based approach to focus on local independence questions. Constraint-based approaches require a statistical test of the conditional dependence and independence in the data. The problem with this approach is that the answer is not very accurate. Mistakes can be made when checking for conditional independencies; for example, maybe A and B are really dependent, but in the data it looks as if they are independent because we do not have enough data (Jensen & Nielsen, 2007). Overall, constraint-based approaches offer the Bayesian network as a representation of independence. This approach tries to find the best network structure to explain the dependencies and independencies by making use of some testing for conditional dependence and independence in the data. The drawback of these approaches is that they can be sensitive to the failure of individual tests for independence (Koller, Friedman, Getoor, & Taskar, 2007).

3.1.2. Score-Based Approach

Score-based approaches are where each possible network structure is assessed by a score, which measures how well the network structure fits the data (Markowetz & Spang, 2007). To start with, this approach defines a search space that consists of a set of possible network structures from the domain of interest. Then, it scores possible network structures using a scoring function. It is necessary to define a search procedure in order to search within the search space and return the network structure with the highest score.

There are three principal issues with using the score-based approach for Bayesian network structure learning: the structure space, the scoring function and the search procedure. In the score based approach, a neighbourhood relation (connectivity) of the search space is typically defined by some operation to move from one point of the search space to another. Generally, the search space results from defining the neighbourhood relation on network structure learning by the shift between neighbourhood structures if they differ by one edge. The difference between neighbouring structures is either the absence of an edge in one of them, adding an edge, or the reversal of an edge.

A scoring function measures how well the network structure fits the data. It is not clear how it finds the highest-scoring network, but it can find the optimal network in some situations. Therefore, the drawback of score-based approaches is that there is no guarantee that they will find the optimal network. Nevertheless, the computational issue is to find the highest-scoring network, as the space of a Bayesian network structure contains a superexponential number of network structures $2^{O(n^2)}$(Koller et al., 2007). In most cases the problem is NP-hard, and we will discuss this in detail in section 3.5.

An example of methods that combine a constraint-based approach and score-based approach is discussed by Tsamardinos, Brown, and Aliferis (2006). They represent the max-min hill-climbing (MMHC) algorithm, which combines both the score-based and constraint-based approach into a single hybrid algorithm. The first phase of the MMHC identifies the parent and children sets of each variable, and then the hill-climbing algorithm is applied. The second phase of the MMHC is used to choose which edges will be in the final network and to orient the edge directions based on the network structure identified in phase one.

3.1.2.1. *Scoring Function*

In Bayesian network structure learning, a scoring function assesses how well a given structure fits the data. Then, it finds the best Bayesian network that maximises this scoring function.

One of the choices for the scoring-based method is the maximum likelihood, which attempts to pick the best network structure that fits the data. The maximum likelihood score is given by:

$$Score\ ML(G) = max_{\theta}\ P(D|G,\theta)$$

This attempts to maximise $P\ (D|G,\theta)$; the likelihood of data conditioned on structure and parameter from the local distribution. The disadvantage of this method is overfitting, as this means that it may not be suitable to choose the best structure. One way to overcome this is by restricting the likelihood with regards to the complexity of structure. Overfitting occurs when a model attempts to fit the data. It is usually a problem in structural learning, as more complex models will offer a better fit to the data compared to simpler models. In statistics, one approach to overfitting avoidance is the use of a penalty that penalises the number of parameters; for example, the number of unknown parameters. However, there is no need for penalties in the Bayesian approach because it penalises the model complexity naturally (Berger et al., 2001).

Therefore, a penalty in the scoring function for the complexity of the Bayesian network structure is needed. (Friedman, K. Murphy, & Russell, 1998) describe a scoring method that is derived from the posterior distribution of a Bayesian network structure. When fitting models, overfitting is more likely to occur because it is possible to increase the likelihood by adding parameters. The Bayesian information criterion (BIC) resolves this drawback by including a penalty term for the number of parameters in the model. However, it can be difficult to state the parameter priors and evaluate the integral $P(D|G)$. Therefore, BIC avoids the integration computation of $P(D|G)$. The BIC score is used to rank possible network structures:

$$BIC\ score = \log \Pr(D \mid G,\ \widehat{\Theta_G}\ ) - \frac{\log N}{2}$$

$\widehat{\Theta_G}$ is the parameter configuration of the network structure G that maximises the likelihood function, where N is the number data instances.

Minimum description length (MDL) is another scoring approach used in Bayesian structure learning, and is based on finding the network structure that gives the best compression of the dataset (Friedman, 1996). The MDL principle helps to avoid overfitting, and the MDL scoring function of network structure B given dataset D is:

$$MDL(B|D) = \frac{1}{2}\log N\ |B| - LL(B\ |D)$$

$|B|$ is the number of the parameters in the network, which denotes the network complexity. $LL(B\ |D)$ is the log-likelihood of the network's structure given data, which is the log probability of the generated data given the network's structure. The first part sums how many bits are needed to encode the network B. $\log N$ is the bits for each parameter. The second part measures how many bits are needed for the representation of D.

$$LL(B\ |D) = N \sum_{i} \sum_{x_i \pi_{x_i}} P'_D\left(x_i, \pi_{x_i}\right) \log\left(\theta_{x_i|\pi_{x_i}}\right)$$

Where $\theta_{x_i|\pi_{x_i}}$ is the parameters for each possible value of x_i, given the value for parents set π_{x_i}.

(Heckerman, 1996) describes the Bayesian-Dirichlet (BD) score (a special case of marginal likelihood), which is based on the following assumptions. The first assumption is that the dataset is a *multinomial sample* determined by parameters Θ_{ijk}, which is the probability of variable i having value k, given the j^{th} configuration of the parents. Another assumption is parameter independence. Given a Bayesian network structure G, then:

- $P(\Theta_G|G) = \prod_{i=1}^{n} P(\Theta_i|G)$

Where the vector of parameters is defined as follows: $\Theta_G$is the parameters of a Bayesian network with underlying DAG. Θ_i is the parameters concerning only the variable x_i in G. Parameters associated with every variable in the network structure are independent; this assumption is called Global parameter independence.

- $P(\Theta_i|G) = \prod_{j=1}^{q_i} P(\Theta_{ij}|G) \;\; for\; i = 1, \dots, n$

The vector of parameters Θ_{ij} is the parameters for variable x_i in G, given the j^{th} configuration of the parents. Parameters associated with every instance of the parents in the network structure are also independent. This assumption is called Local parameter independence.

Also, the *parameter modularity* assumption relies on the assumption that if there are two Bayesian network structures G_1 and G_2, and a variable has the same parents in both graph, then it should have the same distribution of the variable of conditional probabilities.

$P(\Theta_{ij}|G_1) = P(\Theta_{ij}|G_2)$ for $j = 1, \dots, q_i$

Another accepted assumption is that parameters have a *Dirichlet* distribution; "given a network structure G such that $P(G) > 0$, $P(\Theta_{ij}|G_1)$ is Dirichlet for all $\Theta_{ij} \subseteq \Theta_G$" such that the exponents α_{ijk}depending on G satisfy:

$P(\Theta_{ij}|G_1) = c.\prod_k \theta_{ijk}^{\propto_{ijk}-1}$ where c is the normalising constant.

The last assumption is that the data are fully observed. These assumptions are used together to drive $P(G|D)$. Then, the Bayesian Dirichlet (BD) scoring function is defined as follows:

$$p(D|G) \propto \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ij})} \cdot \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + N_{ijk})}{\alpha_{ijk}}$$

α_{ijk} is the Dirichlet exponent of $\theta_{ijk}$from the Dirichlet assumption, and N_{ijk} is the number of cases in the data where variable $x_i = k$ and the configuration of parent $\pi_i = j$, and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$, and $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$.

When combined with the previous assumptions, the likelihood equivalence hypothesis presents the following result. Two network structures G_1 and $G_2$are equivalent if they can encode the same joint probability distributions. Moreover, assuming two network structures G_1 and G_2such that $P(G_1) > 0$ and $P(G_2) > 0$; if G_1 and $G_2$are equivalent then $P(\theta|G_1) = P(\theta|G_2)$. This assumption is called *likelihood equivalence*. All of the previous assumptions are used in likelihood equivalence to derive the BDe (Bayesian Dirichlet likelihood equivalence): where $\alpha_{ijk} = \alpha$. $P(x_i = k, \pi_i = j|G)$ is the number of cases in a dataset where $x_i = k$ and $\pi_i = j$. Here α is the user's equivalent sample size for the $P(\theta|G_2)$.

Similarly to the BD score, the BDe entails knowing $P(x_i = k, \pi_i = j|G)$ for all i, j and k. A particular case of BDe appears when

$$P(x_i = k, \pi_i = j|G) = \frac{1}{r_i \, q_i}$$

The prior network assigns a uniform probability to each configuration of x_i. The resulting score is called BDeu ("u" stands for uniform joint distribution).

Silander, et al. (2007) note that BDeu's marginal likelihood score is commonly used in learning network structures. In order to gain the BDeu score, we need to have the parameter value of α (the equivalent sample size) in order to state the strength of our prior belief in the uniform prior distribution of the network. The authors claim that there is no method to choose the best parameter value for α. They also claim that the obtained network structure is "highly sensitive to the chosen α parameter value" (Silander, Kontkanen, & Myllymäki, 2007). In addition, we get different optimal graphs depending on the value of α, as we do not know the value of α. If we have a prior distribution for the value of α, we can then average over the different possible values of α and choose the best model that way.

Koller, et al. (2007) state that typically "An important property of the scores that affects the efficiency of search is their decomposability. A score is decomposable if we can write the score of a network structure g"

$$score(g : D) = \sum_i FamScore(X_i, Pa_i^g : D)$$

However, if a network structure g is independence-equivalent to another network structure g', then each of the scores is score equivalence.

$$score(g : D) = score(g' : D)$$

3.1.3. The Bayesian Model Averaging Approach

Another approach in structure learning is where many potential network structures are generated, instead of just learning a single network structure. This approach attempts to average all the potential network structures. Basically, we cannot learn a single network structure from data in order to represent different network structures. Bayesian learning enables us to estimate the strength from the data that implies the presence/absence of a particular feature. Thus, we can estimate the posterior probability given the data for some feature $f(G)$ over all possible graphs G; for example, the presence of an edge is likely conditioned on the data.

$P(f|D) = \sum_G f(G)P(G|D)$

Unfortunately, the number of the potential network structures is superexponential $2^{O(n^2 \log n)}$, $where\ n\ is\ the\ number\ of\ variables.$

One of the methods used to overcome and decrease this number is to make a restriction on the network structure G, so that for each node there is a bound K for the number of parents. The next section will discuss the Bayesian model averaging approach further (N. Friedman & Koller, 2003).

3.2 Informative parameter Prior And Non-Informative parameter Prior

Recall that the marginal likelihood $P(D|G)$ is the main component of the Bayesian scoring approach. It is the full structure likelihood averaged over parameters of local probability distributions.

$$P(D|G) = \int_{\Theta} p(D|G,\theta)p(\theta|G)d\Theta$$

Computation of $P(D|G)$ relies on the choice of local probability distributions and local priors in the Bayesian network structure. However, in order to compute $P(D|G)$, the prior $p(\theta|G)$ must correspond to the likelihood $p(D|G,\theta)$. This correspondence is called *conjugacy*. If the posterior probability has the same form as the prior distribution, then this prior distribution is *conjugate* to $p(D|G,\theta)$ (Markowetz & Spang, 2007).

(Cooper & Herskovits, 1992) discuss the marginal likelihood $P(D|G)$ for discrete Bayesian networks. For simplicity, they assume that all network structures are considered equally likely, a priori.

The prior distribution can be represented by a set of possible parameter values, stating our uncertainty about θ. The binomial distribution $Bin(m|N,\theta)$, gives the probability for any number of successes regarding the observation of observing that, for a sequence of n independent trails success/failure and where the data is denoted by m, "we have seen that the uniform prior distribution for θ implies that prior predictive distribution for m (given n) is uniform on the discrete set {0,…,n} given an equal probability to the n+1 possible value" (Gelman, Carlin, Stern, & Rubin, 2003).

It is important to state a prior distribution for θ in a binomial model in order to carry out Bayesian inference. Thus, so far, making the assumption that θ has a *prior uniform distribution* within the intervals [0,1]. Generally, we are uncertain about θ, or know nothing and, therefore, uniform prior is appropriate here. If the posterior

distribution has the same parameter quantity as the prior distribution, then it is called *conjugacy*. One of the benefits of a conjugate prior distribution is that it is suitable for the computational issue of being interpretable extra information.

In spite of the example of population, it is difficult to construct the prior distribution if it has no base in the actual population. Therefore, the prior distribution should have little effect on the posterior distribution, so that the inferences have no influence, and the prior density should be flat or non-informative. However, ignoring useful information is a bad idea, as an *informative prior* expresses certain information about a variable; for example, is a prior distribution for the temperature at tomorrow's temperature.

3.3 Bayesian Structure Learning along with the Approaches to Assigning Priors Information

This section goes into detail about Bayesian structure learning approaches, and follows this up with the approaches that incorporate prior information. A considerable number of search algorithms are used in the search space to obtain the best network structure. A simple and fast, but still powerful, approach to learn structure is a hill climbing algorithm, which explores local moves in the search space. It chooses an initial network structure in the search space to start from: the empty graph in this case. Then, it continually applies a local move to the current network structure by adding an edge, which leads to the best scoring network. This is repeated until no local alteration of the current structure improves the graph score. Finally, if there is no graph in the neighbourhood that has a better score than the current graph, the search procedure stops because the local optimum has been reached. This method finds local maxima of the Bayesian network (Markowetz & Spang, 2007).

Another approach is presented by (Nir Friedman, 1999), who introduced the *Sparse Candidate Algorithm*. Basically, this algorithm obtains a fast performance in learning by restricting the search space. It searches for pairs of nodes that are highly dependent in order to restrict the number of candidate parents for every individual node.

(Koller et al., 2007) discuss the strategies that are used to improve the network returned by a greedy search algorithm. One of these improvements is the random restart: when an algorithm is stuck at a local maximum we restart the search again with different random restarts. As a result of restarting the greedy search, we will eventually discover an optimal network. Another improvement is to avoid all structures in a list of K most visited network structures, this is called a TABU search.

Cooper and Herskovits (1992) propose a search procedure, the *k2 algorithm*, which assumes that there is an ordering on the nodes. The k2 procedure is one of the approaches for maximising $P(G, D)$. As a starting point, a hypothesis is made that there is an ordering on the nodes, and that all the network structures are equally likely. Another hypothesis is also formed: that a node has no parents. Then, we keep adding a parent until no single parent can increase the probability. We shall use the following function:

$$g(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!$$

Where r is the number of values for the variable i. N_{ijk} is the number of instances in dataset D; $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$. However, the assumption is made that the prior probability $P(G)$ can be computed as $P(G) = \prod_{1 \le i \le n} P(\pi_i \longrightarrow x_i)$. Therefore, "for all distinct pairs of variables $x_i$and x_j, our belief about x_i having some set of parents is independent of our belief about $x_j$having some set of parents". Moreover, the probability $P(\pi_i \longrightarrow x_i)$ can be derived by some other method and can also be measured explicitly so that "one method would be to assume that the presence of an arc in $\pi_i \longrightarrow x_i$ is independent of the presence of the other arcs there; if the probability of each arc in $\pi_i \longrightarrow x_i$ is specified, we then can compute $P(\pi_i \longrightarrow x_i)$ " (Cooper & Herskovits, 1992).

(Russell & Norvig, 2010) describe a *Markov Chain Monte Carlo* (MCMC) algorithm. This procedure creates a movement from one state to another, according to a transition probability. In the state space, let $q(x \dashrightarrow x')$ be the probability that a movement is made from one state x to another x'. This transition probability describes the term *Markov chain*. The procedure is repeated until the chain converges to the *stationary distribution*. Assuming that the chain runs for t steps, $\pi_t(x)$ is the probability of being in state x at time t. Let π_{t+1}be the probability of being in state x'at time $t + 1$. We can say that the chain has reached its stationary distribution if $\pi_t = \pi_{t+1}$;

$\pi(x') = \sum_X \pi(x)\, q(x \to x')\ for\ all\ x'$

Friedman and Koller (2003) demonstrate the use of an MCMC algorithm. However, this approach is executed by searching amongst orders of nodes instead of amongst the network structure. They argue that "the space of orders is smaller and more regular than the space of structures, and has a much smoother posterior" (N. Friedman & Koller, 2003). There are $n!$ possible orders; making a uniform prior over orders $\prec$. In addition, a *Markov chain* $\mathcal{M}$ is defined through a space that contains all $n!$ possible orders. The *Markov chain* is designed to ensure it has a *stationary distribution* $P(\prec|D)$. Then, a simulation is made allowing $\mathcal{M}$ to gain a series of orders sampling $\prec_1, \dots, \prec_T$ The *Metropolis algorithm* is used to make sure that the chain is *reversible* so that $P(\prec \to \prec`) = P(\prec` \to \prec)$, and the *stationary distribution* is a posterior distribution $P(\prec|D)$. Furthermore, for every single order $\prec$, we identify the probability which will propose a movement from $\prec to \prec`$ which this function called a *proposal probability* $q(\prec` \mid \prec)$. This movement can be accepted by a probability:

$$\min\left[\frac{P(\prec`|D)\, q(\prec|\prec`)}{P(\prec|D)\, q(\prec`|\prec)}, 1\right]$$

Furthermore, the authors claim that "the Markov chain over orders mixes much faster and more reliably than the chain over network structures" (N. Friedman & Koller, 2003).

It is possible to conduct MCMC directly over structures, as described by (Koller & Friedman, 2009). This is done by defining a Markov chain using a space of potential network structures whose *stationary distribution* is the posterior distribution $P(G|D)$. In this approach, a set of possible network structures is generated by performing some random walks in this Markov chain, and this is repeated until it reaches its *stationary distribution*. The algorithm considers some local operations to move from one structure to another: these operations are adding, deleting, or reversing an edge. The *Metropolis algorithm* accepting procedure is used; in which the movement can be accepted by a probability:

$$\min\left[\frac{P(G`,D)\, q(G` \to G)}{P(G,D)\, q(G \to G`)}, 1\right]$$

However, there are some problems that possibly limit its efficiency for large domains; for example, including many variables, as the space of the network grows superexponentially in this situation.

Friedman and Koller (2003) described the effect of a structure prior by testing the sensitivity of a Bayesian model averaging approach on the choices of the prior. They compared the results between using a prior over structure—MCMC $P(G)$ and using a prior over ordering—MCMC $P(\prec)$. Friedman and Koller defined a uniform prior over ordering $P(\prec)$ and also need to define $P(G|\prec)$. Every graph consistent with a particular order is equally likely. For example, the empty graph is consistent with all orders: $P(Empty\ graph) = \sum_{\prec} P(Empty\ graph|\prec)$.

The prior over possible structure requires restricting the bound of the number of possible parents, so that a node X_i has K parents; therefore, there are $\binom{n-1}{k}$potential parent sets. In addition, assuming a uniform prior:

$$P(G) \propto \prod_{i=1}^{n} \binom{n-1}{|Pa_G(X_i)|}^{-1}$$

However, with uniform prior over orders, sparse graphs have prior probability than with uniform prior on structure.

The results of a structure learning algorithm are sensitive to the structure prior, and priors can lead to very different results. Moreover, the authors claim that "Given that the choice of prior is often somewhat arbitrary, there is no reason to assume that our order-based prior is less reasonable than any other" (Friedman & Koller, 2003). Network structure is consistent, in that more orderings are more likely. Priors over network structures are used for practical purposes and are easy to work with and simple.

A number of methods and applications have been presented to address relationship identification problems. (Sheehan & Egeland, 2007) show how prior information can be incorporated into this problem. In both human and non-human populations, reconstructing the pedigree of related families and the amount of inbreeding from genetic data is important within a species. An example of the problem of reconstructing pedigrees is found in mass-grave tragedies in which the remains of many individuals are found and can only be recognised by DNA. If *G* is the pedigree Bayesian network structure that contains a set of nodes *V* and a directed edge set *E*, then each node in the pedigree Bayesian network structure represents the genotype of an individual and has one of three possible parent configurations:

- An individual that has only one parent identified in the pedigree, if there is one incoming arrow.
- An individual that has two parents identified in the pedigree, if there are two incoming arrows.
- If there are no incoming arrows, the individual is a founder in the pedigree.

A Bayesian approach is used to include prior knowledge, which is expressed by assigning a prior probability over the space of pedigree samples. The likelihood function here is computed based on DNA datasets for every single pedigree, and this data is updated along with prior probability in order to discover the posterior distribution using Bayes' theorem. Basically, prior knowledge helped rule out some possibilities. They also demonstrate the difference between *hard* and *soft* prior information. Hard prior information is a piece of information that the user definitely knows, whilst soft prior information is where the user writes down some probabilistic equation. However, the *global* prior information in the relationship identification problem is the general knowledge about the population (for example, information about mating behaviour), and the *local* prior information is that related to particular parts of the pedigree. Therefore, hard, local and global information is combined to rule out a number of possibilities. Afterwards, the prior function $\Pr(g)$ is used to assign a prior probability to every pedigree in the sample space for n individuals:

$$\Pr(g) = c \prod_{i=1}^{s} M_i^{b_i(g)} \prod_{\substack{i,k=1 \\ j \neq k}}^{n} R_{jk}^{o_{jk}(g)} \quad where\ c\ is\ the\ normalisation\ constant$$

Where $M_1, \ldots, M_s$ are the global parameters that enable pedigrees based on s, particular information to be weighted. If M_i set to:

$$M_i = \begin{cases} 0, & if\ \ b_i(g) > 0 \\ 1, & otherwise \end{cases}$$

Where $b_i(g)$ is the integer exponent, which corresponds to M_i that offers a specific measure of an individual pedigree g, and, therefore, provides the degree of the relative weightings of different pedigrees for the i individual. Just pedigrees with $b_i(g) = 0$ are accepted (similar to setting $M_i = 0$). Whilst if $M_i = 1$, then this amounts to locating a flat prior. Assigning values between 0 and 1 will increase the probability.

R_{jk} is the local parameters. Setting $R_{jk} = 0$ will exclude the pedigrees presenting j from being a parent of k. If $R_{jk} > 1$, then this will favour a pedigree with specific characteristics. However, there is no general form for choosing the values for R and M. The decision for choosing them is sensitive and should be investigated for a specific domain. The prior function $\Pr(g)$ is simple to interpret and modify.

Angelopoulos and Cussens (2005) discuss a technique for specifying informative priors applied to classification and regression tree (C&RT) models. Basically, C&RT is a method used to classify data into different classes. The authors decided to use a Bayesian approach, so that there is a prior for every possible tree. Then, they used an approximate sample for the observed data. Bayes' theorem is used to approximate the posterior distribution of the trees. The authors state that "the goal of including prior knowledge is to improve decision making under uncertainty" (Angelopoulos & Cussens, 2005). However, the drawback of assigning a prior to each possible tree is that this is very difficult when you have a large number of possible trees. So, in this particular case they took a novel approach to defining a prior using a sampling algorithm, a *stochastic logic program* (SLP). An SLP can be used to define a prior over a given space of statistical models. In addition to this, the SLP will generate a tree each time, which specifies the prior implicitly. This approach is proposed instead of using a *closed-form expression*, which defines a prior by writing down some equation. They used an MCMC algorithm to take approximate samples from the posterior probability over all C&RT models. The MCMC proposal distribution is based on the prior.

The same SLP approach can be used to effect a Bayesian approach to Bayesian network learning (Angelopoulos & Cussens, 2009). Once the model spaces have been defined using logic programs, then the SLP is used to define an informative prior on the Bayesian network structure over model spaces. They demonstrate that learning Bayesian networks with priors achieves a robust result, and an informative prior increases the quality of the results a lot. The way hard information is represented in a prior distribution is by reducing some network structures to zero probability, which sets the posterior probability to zero regardless of the data. Also, they discuss the important use of Markov equivalence classes for setting priors. Moreover, the authors

also claim that "If we consider working in a model space of Markov equivalence classes there is no inconsistency: the prior probability of a Markov equivalence class of BNs can be defined to be the sum of the priors of the BNs in that class. There seems no reason why these Bayesian network priors need be equal. Of course, it may be more convenient to define a prior directly on the Markov equivalence class: this is a knowledge engineering issue" (Angelopoulos & Cussens, 2009).

However, there are often huge practical difficulties with the application of Bayesian analysis. Moreover, formalising prior knowledge requires a representation language in order to bridge the gap between the prior information in an individual's brain and that stated by a probability distribution. Also, one of the difficulties with the Bayesian approach is getting hold of the posterior. Therefore, a conjugate prior distribution is typically used to simplify the computation of the posterior distribution and its representation.

(Castelo & Siebes, 1998) discuss how a user can assign a prior probability to each DAG. In this approach, a user has to specify partial prior knowledge, which is completed later to create full prior knowledge over all possible Bayesian networks. A degree of belief over the dependency between two variables is coupled with the nature of the models they try to induce. The approximation of the full prior knowledge is done using *directed graphs*. They assume that the user's prior belief is coherent. Moreover, the user's prior knowledge over the three possible links between two variables must obey a probability distribution. For example, let a and b be nodes in a Bayesian network and let $P(a \rightarrow b)$be the probability for an edge, $P(a \leftarrow b)$ be the probability for the other direction, and P($a \ldots b$)be the probability that there is no edge; then:

$P(a \rightarrow b) + P(a \leftarrow b) +P(a \ldots b) = 1$

The priors for all the potential network structures sum to one. However, when considering two options for a normalising constant, the distribution would then be:

$$P(g) = c + \prod_{\substack{v_{i,}, v_j \in V \\ i \neq j}} P\left(v_{i,} \leftrightharpoons v_j\right)$$

Or

$$P(g) = c \prod_{\substack{v_{i,}, v_j \in V \\ i \neq j}} P(v_{i,} \leftrightharpoons v_j)$$

Also, (Koller & Friedman, 2009) illustrate the advantages of making the assumption that a structure prior satisfies *structure modularity*, where the prior structure $P(g)$ is proportional to multiplying the terms $(Pa_{X_i} = Pa_{X_i}^g)$, which represent locating the prior probability to choose a set of parents X_i. It is represented as:

$P(g) \propto \prod_i P(Pa_{X_i} = Pa_{X_i}^g)$

(Buntine, 1991) demonstrates the use of incorporating expert knowledge into learning, and how the information is converted into a prior on Bayesian networks. The author assumes that we know the variable ordering and that we have expert knowledge E. The expert specifies the total ordering $\prec$ for the variables, in which the parents of a variable must be less than the variable. In other words, for instance, if $y \in \pi_x$ then $y \prec x$. If we have $x, y \in X$, in which $y \prec x$, then the prior probability of y being a parent of x is represented as $\Pr(\pi| \prec, E)$. As we have the ordering, the prior probability for a particular graph is:

$$\Pr(\pi| \prec, E) = \prod_{x \in X} \Pr(\pi_x| \prec, E),$$

$$assuming\ \pi\ is\ consistent\ with \prec, where$$

$$\Pr(\pi_x| \prec, E) = \left(\prod_{y \in \pi_x} \Pr(y \to x| \prec, E) \right) . \left(\prod_{y \notin \pi_x} \Pr(y \to x| \prec, E) \right)$$

In addition, the expert specifies the total number of variables $\prec$. Along with a prior probability for every possible parents set E, this combined information is represented as *Sample*; $\Pr(\pi|Sample\ , \prec, E)$. The structure posterior is computed as:

$$\Pr(\pi|Sample\ , \prec, E) = \sum_{\pi_x \in P_x \wedge y \in \pi_x} \Pr(\pi_x|Sample\ , \prec, E)$$

Where P_x is parent structure.

(Tsamardinos, Brown, & Aliferis, 2006) also discuss the max-min hill-climbing (MMHC) algorithm, which combines both the score-based and constraint-based approach into a single hybrid algorithm. Conditional independence is assessed using a statistical test on the data. This test assumes independence and does not consider the null hypothesis when two variables are conditionally dependent. The first phase of the MMHC identifies the parents and children set of each variable, and then the hill-climbing algorithm is applied. The second phase of the MMHC is used to choose which edges will be in the final network, and also to orient the edge directions based on the network structure identified in phase one.

Another approach to incorporating prior information is introduced by (Borboudakis & Tsamardinos, 2012), who present algorithms for incorporating path constraints to *Partially Directed Acyclic Graphs* (PDAGs) and *Partially Oriented Ancestral Graphs* (PAGs). This path constraint is about the presence or absence of (possibly indirect) causal relations in a causal model. Moreover, the incorporation of causal knowledge into a PDAG (PAG) forces the orientation of certain edges, which results in a corresponding PC-PDAG with fewer structural uncertainties.

Also, (Campos, Zeng, & Ji, 2009) present a novel algorithm for the exact learning of Bayesian network structure from data that incorporates an expert's knowledge, which is based on (decomposable) score functions. It combines structural and parameter constraints with data through a branch-and- bound (B&B) approach to ensure global optimality with respect to the score function.

(Mansinghka, Kemp, Tenenbaum, & Griffiths, 2006) provide a Bayesian hierarchical framework that incorporates edge priors, using MCMC for sampling networks, and therefore improves the structure of the network recovery. Their approach is built on nonparametric, hierarchical Bayesian models that finds graph regularities in terms of node classes. Meanwhile, (Werhli & Husmeier, 2007)), used a different form of incorporating prior knowledge using MCMC for sampling networks. A Bayesian approach is adopted to incorporate various sources of prior knowledge in terms of an energy function. From this function a prior distribution over structures is found in the form of a Gibbs distribution, from which the penalty on a particular edge can be determined.

3.4 Existing Tools for Bayesian Network Learning

In this section, the existing tools to learn Bayesian networks from the data are examined. Kevin Murphy's website lists a lot of software packages for graphical models. I went through most of them and found that some of them work fine, whilst others do not (as a result of broken links, etc.). This section gives an overview of some of the existing tools that I checked.

To begin with, UnBBayes (k2 for structure learning) is a probabilistic network framework with a graphical user interface to enable a user to perform sampling, learning, and evaluation. UnBBayes supports Bayesian networks, structures, and parameters, and shows the efficacy of probabilistic reasoning. It represents user degrees of certainty; for example, it can predict whether a statement is more likely to be true or false. UnBBayes is a flexible tool that enables users to manipulate and build a Bayesian network based on a knowledge domain. The software allows the creation of a Bayesian network from scratch, which also allows the user to add nodes and edges, and edit CPTs. Moreover, UnBBayes uses a junction tree algorithm to perform Bayesian inferences.

GOBNILP (Globally Optimal Bayesian Network learning using Integer Linear Programming) learns Bayesian networks from complete discrete data or from local scores. GOBNILP is a free, publicly available Bayesian network structure-learning package. Also, GOBNILP can find the optimal network, given a constraint on the maximum number of parents, which is 3 by default. In this research, GOBNILP is used to find the optimal networks, and in some experiments prior knowledge is consistent with the optimal network.

Banjo is software for static and dynamic Bayesian structure learning. It performs structural inference in Bayesian networks using a BDe score for discrete variables. The search procedure is based on simulated annealing and greedy algorithms. A search algorithm in Banjo consists of a set of main components that suggest a new network, or number of networks, and then checks the suggested networks for cycles,

computes the score of the suggested network, and determines whether the suggested network is accepted or not.

B-Course, which is represented by U. Helsinki, is a *web application tool for Bayesian modelling*. The software has two main options. The first consists of dependency models that allow a user to go through clear steps of Bayesian modelling and inference. The second option is classification modelling, which uses Naive Bayes Networks. A user can either use their own data, or example datasets provided by the website. The network is run on B-Course's server and the results are viewed on their website. They use both a simple and a greedy random search, and a BDe score.

Finally, bnlearn is an R package that includes several algorithms for Bayesian networks structure learning, with either discrete or continuous variables. Furthermore, bnlearn also allows users to use either a constraint-based approach or score-based algorithms with different scoring functions. A user can incorporate prior information in the data by means of the blacklist and whitelist arguments.

3.5 The Complexity of Bayesian Network Learning

This section discusses the theoretical limits of learning Bayesian networks. Learning the structure of a Bayesian network is an NP-hard problem. It is difficult to search all possible Bayesian networks for different sets of variables. The number of graphs grows exponentially with the number of variables. This problem also exists with a small number of variables, as there are many DAGs to consider in the search space and it is difficult to search for a high scoring network that fits with the data.

(Cormen, Leiserson, Rivest, & Stein, 2009) explain computational complexity. Consider a particular algorithm which takes the graph that is input and checks whether there is a *Hamiltonian cycle* or not, and returns a yes/no answer. The Hamiltonian cycle of an undirected graph is a simple cycle that contains each vertex. The *Hamiltonian* cycle problem can be defined as "Does graph G have a Hamiltonian cycle?". A graph that does have a *Hamiltonian* cycle is called *Hamiltonian*, and if not then it is called *nonhamiltonian*. So the formal language is represented as:

$HAM - CYCLE = \{< \mathrm{G} > : \mathrm{G} \text{ is a Hamiltonian graph}\}$

The algorithm is difficult because deciding whether there is a *Hamiltonian* cycle or not takes a long time. Algorithms that are NP are those that give you a yes/no solution, which is known as a *decision problem*. Generally, the length of time it takes for an algorithm to run depends on how big the input is. In Bayesian learning, we are trying to find the highest score in a Bayesian network. So, in cases where the dataset consists of two variables, the algorithm runs very quickly. However, as the number of the variables increases, the problem gets harder. A Polynomial time algorithm has a run time based on an input of size n for the worst situation running time $O(n^k)$for some constant k. However, not all problems can be solved in Polynomial time.

In addition, if an algorithm takes a Polynomial time to finish, it would appear to be a quick algorithm. On the other hand, if we are using an algorithm where the answer is yes/no, and the *Hamiltonian* for checking a possible solution and whether there is a correct solution that will take a Polynomial amount of time, then the problem is

known as an NP-problem. An NP-problem is one way of reaching a decision in cases when the time to solve the original problem could be very long. However, checking for verification is easy; this is known as *Polynomial Time Verification*. In addition, "the complexity class NP is the class of languages that can be verified by a Polynomial time algorithm. More precisely, a language L belongs to NP if and only if there exists a two-input Polynomial time algorithm A and constant c such that:

$$L = \{x \in \{0,1\}: there\ exists\ a\ certificate\ y\ with\ |y| = O(|X|^c)$$

We say that algorithm A verifies language L in Polynomial time" (Cormen et al., 2009). Therefore, $HAM - CYCLE \in NP$ if there is a Polynomial time algorithm to choose L. This algorithm can easily be transformed to a two argument input verification algorithm by accepting the exact string that is defined in L. Therefore, $P \subseteq NP$. Where P problems are considered easy to solve, NP problems are easy to check.

According to Cooper (1990), the computation of the probability of a particular variable of interest, given other variables, is called *probabilistic inference*. The author assumes that the nodes represent propositional variable Y, which would have an assigned value of either true (T) or false (F). The *probabilistic inference* of a Bayesian network is used to mean the computation of $P(S_1|S_2)$, where $S_1$is either a single assigned value or a combination assigned value. While S_2 is the combination assigned value, and the computation of the probabilistic inference in the case that no explicit conditioned information ($Y = T$) is NP-hard. Furthermore, Cooper illustrates that *probabilistic inference* using Bayesian networks is NP-hard. Thus, "it seems unlikely that an exact algorithm can be developed to perform probabilistic inference efficiently over all classes of belief networks" (Cooper, 1990).

(David Chickering, Heckerman, & Meek, 2003) explain the problem of finding the best Bayesian network structure in which each node has at most k parents, for $k \geq 3$. Therefore, finding the highest scoring network structure is NP-hard. However, many Bayesian network learning algorithms do not guarantee to return the high scoring network.

As discussed earlier, a number of approaches for learning a Bayesian network are based on two main elements. One of these is defining the score metric, such as the BDe metric, which calculates a score for each possible network structure. These scores indicate how the network structure fits with the data. The other element is that the search procedure attempts to yield the highest score amongst all the possible network structures that are computed by the score matrix (D. Chickering, Geiger, & Heckerman, 1995). However, the authors discuss the problem of finding the highest scoring network from all the possible network structures so that every node has no more than k parents. Consequently, the main drawback is where every node has at most k parents; which is NP-hard for $k > 1$. For the general case K-LEARN:

"INSTANCE: Set of variables U, database $D = \{C_1, \dots, C_m\}$,, where each C_i is an instance of all variables in U, scoring metric $M(D, G)$ and real value p".

"QUESTION: does there exist a network structure G defined over the variables in U, where each node in G has at most k parents, such that $M(D, G) \geq p$?" (D. Chickering et al., 1995).

In addition, one of the approaches demonstrates that K-LEARN is NP-complete for $k > 1$ in the case when using BD metric.

Algorithms are used to solve a number of problems that also have different complexities (Koller & Friedman, 2009). In optimisation problems, the target is to maximise a problem for a potential solution σ in a given solution space $\sum$. Additionally, the evaluation of the value of every possible solution is done by an objective function $f: \sum \rightarrow \mathbb{R}$. The main target is to obtain a solution that yields the maximum score:

$$\sigma^* = \arg max_{\sigma \in \sum} f(\sigma)$$

In an optimisation problem, where the solution space contains discrete hypothesis, the number grows exponentially in most cases, related to the size of the problem for the number of the solution space $\sum$. Therefore, we cannot enumerate in order to obtain the best solution.

A decision problem is a problem with a "yes" or "no" answer. The complexities of decision problems are basically harder than those that can be answered by a *nondeterministic Turing machine* in polynomial time. When a decision about a combinatorial optimisation problem is shown to NP-complete problems, the subsequent optimisation is NP-hard (Atallah & Blanton, 2009).

In this chapter, the problems of learning both parameters and structures that are also based on fully observed data were considered. The three main approaches for Bayesian structure learning—constraint-based, score-based, and the Bayesian model averaging approach—were discussed. This chapter demonstrated why it is important to study prior knowledge and reviewed the several approaches that have been considered. Different related work on Bayesian structure learning approaches were highlighted and some existing applications for Bayesian network learning were examined. Finally, the theoretical limits of learning Bayesian networks were presented.

.

4 Learning algorithms that use prior knowledge

This chapter presents an algorithm to learn Bayesian networks from data. It demonstrates Cowell's (2009) approach to the exact learning of the maximum likelihood Bayesian network. Then, it goes into detail about the developed heuristic search-learning algorithm (based on a hill-climbing algorithm) and shows how the directed acyclic graph (DAG) is represented. It also describes how cycle checking and the scoring function are used in the developed learning algorithm. Finally, it presents a detailed discussion about the search procedure in the hill-climbing algorithm.

4.1 Introduction

Finding the Bayesian network that maximises a score function is described as structure learning. Most work about Bayesian network learning has focused on heuristic searches, where there is no guarantee that the optimal network will be found. However, there is an increasing trend towards the work on exact Bayesian network structure learning. One approach is to use dynamic programming, which has been used successfully, as long as there is a limit on parent set sizes. This chapter aims to develop a Bayesian network learning algorithm that can be used to incorporate prior information. Overall, there are three approaches: the score-based approach, the constraint-based approach and the model-averaging approach. People typically use a non-Bayesian method in the constraint-based approach, which requires a statistical test of the conditional dependence and independence in the data. Another reason why the constraint-based approach does not normally use the Bayesian approach is because it uses hard information. The model-averaging approach considers several possible networks. Therefore, it is not possible to choose the best network, as you cannot be sure which is the correct one.

Some Bayesian learning algorithms are heuristic search algorithms, which attempt to maximise a score using some data. Our current experimental work uses a hill-climbing algorithm (a heuristic search) that makes local moves, which lead to a local score-maximal Bayesian network. Score-based approaches employ different scores, and devote the most effort to maximising scores without much consideration of prior information. This thesis investigates the score-based approach.

The main reason for working on this exact dynamic programming algorithm is because it is commonly used. It is also an exact learning approach that always finds the optimal network, which makes it easy to validate whether things work. Nevertheless, it is not clear what else can be done to incorporate any prior knowledge when using the exact dynamic programming dynamic programming algorithm, so I conducted research regarding this question. There is some motivation to extend the exact dynamic programming (2009) approach towards exact learning in order to incorporate different sorts of prior knowledge and investigate the effect of the prior knowledge on the dynamic programming algorithm.

The structure learning of Bayesian networks is an NP-Hard optimisation problem since the number of structures grows exponentially with the number of variables. As learning Bayesian networks are NP-hard and these exact learning approaches will not scale to bigger datasets, exact leaning approaches are not the answer to every problem due to scalability issues. Thus, we have to use a greedy approach, such as the hill-climbing approach. Consequently, we need to explore improvements to the hill-climbing approach. The high cardinality of the search spaces of heuristic approaches have been shown to be effective and efficient; therefore, I researched this question. As a result, there is some motivation for designing an algorithm that uses prior knowledge as input data, while simultaneously dealing with bigger problems.

4.2 Dynamic Programming

As a starting point for this thesis, I used Cowell's (2009) approach to the exact learning of the maximum likelihood Bayesian network. The main reason for working on this dynamic programming algorithm is because it is commonly used. It is also an exact learning approach that always finds the optimal network, which makes it easy to check whether things work. However, it is not clear what else can be done to incorporate any prior knowledge when using the exact dynamic programming algorithm. A user provides some prior information, in addition to the dataset, and the goal is to find the most likely network that fits the user's prior knowledge.

(Cowell, 2009) demonstrates that reconstructing the pedigree of related families using genotype data is an important task. Learning a pedigree is like learning Bayesian networks from data, as each node denotes the genotype of an individual. Cowell uses an algorithm (proposed by Silander & Myllymäki, 2006) to obtain a maximum likelihood pedigree using fully observed data gained from the genotype data of related family samples.

This exact dynamic programming is an attempt to search for a pedigree of up to 31 individuals (approximately). Given a pedigree on n nodes in a set V the dynamic programming algorithm approach is used to find the set of local scores of possible parent configurations for each individual i. This approach tries to find the best sinks for any subset of variables, where sinks are variables that have no children. The basic idea here is to find the sinks for bigger subsets by using the sinks for small subsets. Then, it identifies the best ordering of the best sinks. The dynamic programming attempts to find the best sink i for V, and subsequently the best sink for $V\backslash\{i\}$ and so on. In this way, we can find the best sink for the entire set of all vertices. The entire best parent set for that sink is the one with highest scores. Any Bayesian network has one or more total ordering. The last variable in that total ordering will always has the best possible parent set to choose from, which has a high score.

The algorithm consists of four main steps. The first step of the reconstruction technique is where we search a list Λ_i for the possible set of parent combinations (j, k) for each individual i ∈ V. Every Λ_i for the corresponding local scores for parent configurations α(i|j, k) is sorted in decreasing order:

- Find the Local scores $\alpha(i|j,k)$
- Order the local scores in a decreasing order.

In the second step, we find the best score and the best sink for each subset of V. To begin with, we need to generate a subset W from the set of all variables V so that for every subset there is a score and a sink. If we want to get the score for a particular subset, we go through each member of $\boldsymbol{i}$ and remove it to get $\boldsymbol{U}$ then we find the best parent sets for $\boldsymbol{i}$. We make sure that we compute this beforehand, as this is just a looking-up procedure.

- Generates W from V
- For every $W \subseteq V$ in order Do
  - $Scores[W] \leftarrow 0.0$
  - $sinks[W] \leftarrow -1$
- Loop for all $i \in W$ Do
  - $U \leftarrow W \setminus \{i\}$
  - $skore \leftarrow BLS(i,U) + scores[U]$

    $// \; BLS \; is$ best Local Score
    - $if \; (\; sinks[W] = -1 \; or \; skore > scores[W] \;) \; Do$
      - $Scores[W] \leftarrow skore$
      - $Sinks[W] \leftarrow i$

In the third step, if element i was a sink for W, then we look for the parent set that will work best.

- $left = |V|$
- For i=$|V|$ to 1 Do
  - $ord[i] \leftarrow sinks[left]$
  - $left \leftarrow left \setminus \{ord[i]\}$

Finally, we simply recall the best sink and score for each subset.

- $predecs \leftarrow \emptyset$
- *For i=1 to 1 to |V| DO*
  - $parent[ord[i]] \leftarrow Bpset(ord[i], predes)$

    // Bpset is the best parent sets
  - $predecs \leftarrow predecs \cup \{ord[i]\}$

4.3 A Hill Climbing Algorithm with Prior Knowledge (HCPK)

A simple and fast approach in a heuristic search for structure learning is hill climbing, which explores local moves in the search space. It chooses an initial network structure in a search space to start from; an empty graph was used in this research. Then, the algorithm continually applies a local move to the current network structure by adding an edge, if that leads to a better score. This is repeated until no local moves applied to the current structure improve the graph score. Finally, if there is no graph in the neighbourhood that has a better score than the current graph, the search procedure stops because a local optimum has been reached. This method finds local maxima of the Bayesian scoring metric. In this research, we aimed to keep the hill-climbing as simple as possible. If we start with empty graph, then it is possible to construct any Bayesian network by adding edges. In this research, we explored the difference between this particular algorithm and the same algorithm with prior knowledge.

4.3.1 Representation of DAG

Recall that the structure of a Bayesian network is a directed acyclic graph (DAG), in which nodes represent the random variables and edges represent probabilistic dependence among variables. Here, a DAG is represented by specifying the parent set for each vertex; for example, if variables B and C are parents for child A, this is represented as $A \leftarrow [B, C]$.

4.3.2 Cycle Checking

Since DAGs are acyclic, each time our hill-climbing algorithm makes a change it must be checked for a cycle. A list of current ancestors for each node is maintained, which allows for the fast checking of cycles.

4.3.3 BDe Score

There are a number of different methods of scoring. In this learning algorithm, the Bayesian Dirichlet likelihood equivalence (BDe) scoring method is used. Each variable in a BDe score is reviewed in order to discover which parents it has in the graph. The BDe score is the (marginal) probability of the observed data conditional on the graph structure, assuming a Dirichlet parameter prior (Heckerman & Chickering, 1995).

4.3.4 Search Procedure

The local moves in this algorithm are the addition of an edge. In this HCPK, we chose an empty graph as a starting point in the search space. Basically, the algorithm computes the local BDe Score for each child and its possible parent sets, and chooses the one with the highest score. This process is repeated until no graph has a larger score than the current graph. The details are as follows (Algorithm 1):

The algorithm shown takes all of the variables $\mathrm{V} = \{x_1, x_2, \dots, x_n\}$ from the dataset D and places them in an arbitrary order V. Each child x_i is selected from the ordered set V. The remainder of the variables, not just those who are earlier in the order, are the possible parents $\mathrm{PP}_a(\mathrm{x})$, and can be added as parents in any move. The objective is to examine each variable and find the best possible parent sets.

When choosing a parent set for x_i, the algorithm works by adding one parent $\mathrm{P}_a(\mathrm{x})$ at a time. The number of possible parents that may be added is limited to a certain, adjustable number. Therefore, the user needs to set a parameter in order to establish that limit.

The variable earliest in the order has the best possible parent sets to choose from, whilst it is more difficult to arrive at a good parent set for the variables later in the order because it is harder to avoid cycles.

The algorithm computes the local score of the possible parent sets for each child. Since the BDe score is *decomposable* into local scores, this is all that needs to be computed after each local move. The algorithm performs some checks, as there are some sequences concerning the early decisions that the algorithm needs to make if we do not have prior knowledge.

Before computing the local score for the possible parents, we have to check that the current possible parent does not violate any constraints at line 11. If X has been previously chosen as a parent of Y, then Y will be excluded when we choose a parent for X.

The algorithm considers all of the possible sets and chooses the one with highest score. In addition, if any given local score is better than the best score, the set is assigned to the best score. Then, the algorithm adds edges from parents to children and updates the ancestor relation. Since DAGs are acyclic, each time our HCPK algorithm makes a change it must be checked for a cycle, which entails an additional step. A list of current ancestors for each node is maintained, which allows for the fast checking of cycles. However, if there is prior knowledge, Algorithm 1 checks whether this is satisfied by using an extra check at line 11.

Algorithm 1: HCPK algorithm

```
1.  V ← [allVariables]
2.  g ← empty
3.  For i = 0 to length(V) Do
4.      BP_a(x) ← [ ]  // Best Parents Combination
5.      x ← V[i] // child
6.      PP_a(x) ← V − x // Possible parents
7.      bestLocalScore ← Score(x, BP_a(x))
8.      For  j = 1 to length(PP_a(x))  Do
9.          P_a(x) ← PP_a(x)[j]
10.         BP_a(x). add[P_a(x)]
11.         IF CheckPriorKnowledge(P_a(x)) == FALSE
12.             Continue
13.         END IF
14.         localScore  ← Score(x, BP_a(x)) {Using BDe score}
15.     IF bestLocalScore  <  localScore AND checkCyclicMap(g , x, BP_a(x)) Then
16.         bestLocalScore  ← localScore
17.         g. update(x, BP_a(x))
18.         {See explanation in section 4.3.2}
19.     ELSE
20.         BP_a(x). Remove[P_a(x)]
21.     END IF
22.     END FOR
23. END FOR
24. Return g
```

4.3.5 Hill Climbing with Random Restart

This algorithm cannot guarantee that it will find the optimal network because it may become stuck at a local maximum, of which there could be many. One run is not enough in HCPK. In this situation, it is best to start the search again. Therefore, we run HCPK several times with different random orderings V of the variables to partially compensate for the myopia of hill climbing. Different random orderings give different scores. Hence, the variable earliest in the order has the best possible parent set to choose from, whilst it is more difficult to arrive at a good parent set for the last variable in the ordering. Many choices of high-scoring parent sets will not be possible, as choosing them would lead to a cycle.

However, the best total score of the ordering is kept. If a new iteration of HCPK produces a better total score, the new score replaces the previous best total score and returns the network with the highest score.

Algorithm 2: HCPK algorithm with random restarts

```
1.    i ← 0
2.    V ← [allVariables]
3.    BestScore ← −∞
4.    While(i < NumberRuns)
5.        V ← RandomOrdering(V)
6.        BN ← HillClimbing
7.        If ( Score(BN) < BestScore)
8.            BestScore ← Score(BN)
9.    i + +
```

This chapter proposed an algorithm to learn Bayesian networks from data. It presented the exact learning of the maximum likelihood Bayesian network. Also, it presented the developed learning algorithm, which is a hill-climbing algorithm prior knowledge (HCPK). Finally, it gave a detailed discussion of the search procedure in the HCPK. While, in the following chapter show how can we incorporate different types of prior to these developed algorithms.

5 Incorporating Prior Knowledge

This chapter introduces the main contribution of this thesis; presenting an algorithm that can incorporate different types of prior into the developed algorithm. This section discusses prior information, and highlights the differences between hard and soft prior information. It then describes how different sorts of prior knowledge are incorporated into the developed learning algorithms.

5.1 Introduction

Prior information is any information people have, in addition to the data, that helps to obtain a good model. Hard information reduces the probability of some network structures to zero. In other words, in order to use prior information you have to take hard information and (perhaps implicitly) express it in the prior distribution to give certain networks zero probability. If the prior probability is zero then the posterior probability will still be zero, no matter what sort of data is available. In contrast, soft prior information gives nonzero prior probability to some possible network structures, and some will achieve higher probabilities than others. However, even if we set a prior probability to a small number, if there is enough supporting data then the posterior probability could be large. Uniform prior knowledge can be used if we do not have any information because it represents a lack of information, which means that each structure has an equal prior probability of being true.

The learning algorithm is intended to enable users to express their knowledge of a variety of problems in a straightforward manner. The main objective of this section is to investigate whether incorporating prior knowledge leads to significantly better results, and to evaluate its effect on learning speed. This section discusses the many experiments conducted during the development process in order to generate numerous results. In this work, we aimed to develop learning algorithms using different datasets and types of prior knowledge. There are many types of prior knowledge, including the

knowledge of whether or not node A is a parent of node B and known topological ordering. The most challenging type of prior knowledge (and the main subject of this research) is known ancestor relations and conditional independence. However, the main issue is that more complicated prior knowledge cannot be incorporated into local score. In addition to this, once we have complicated prior knowledge, simply using hill climbing without changing it will fail because it will constantly generate networks that are not allowed. Therefore, we need to add some intelligence.

5.2 Dynamic Programming

Looking back at the dynamic programming algorithm described in section 4.2. Here, we extend Cowell's (2009) approach towards the exact learning in order to incorporate different sorts of prior knowledge. The exact dynamic programming dynamic programming algorithm is an exact learning approach that always finds the optimal network, and this makes it useful for checking whether things work. In this section, we investigate the effect of the prior knowledge on the dynamic programming algorithm.

There are a lot of different types of prior knowledge, which might be, for example, A has to be the parent of B, or A must not be the parent of B. For this reason, we allow the user to say that a certain arrow has to be there. Another type of prior knowledge is the statement that A has to be an ancestor of B (B cannot be a descendent). Therefore, the algorithm incorporates different types of users' prior knowledge and drops the assumption that there are two parents at most as follows:

5.2.1 Arrows which must be absent $A \nleftarrow B$

The user can specify the prior knowledge that variable 2 must not be a parent of 0. This is a hard constraint, and it is straightforward to incorporate this prior knowledge by removing all the choices of the parents' set from the datasets. Thus, ruling out the parents' set does not fit with this prior knowledge. Moreover, each variable has a choice of parents from the dataset and one that does not fit with the prior knowledge can be ruled out.

5.2.2 Arrows which have to be there $A \leftarrow B$

Given a Bayesian network on n nodes in a set V, for each individual i ∈ V, we search for the valid set of parent combinations in a list Λ_i. We also find the corresponding local scores for parent configuration. Moreover, in the exact dynamic programming algorithm each list Λ_i always has at least one element corresponding to having no parents, and it treats i as a founder. However, if we have prior knowledge, this will rule out the choice of no parents and, as a result, will produce a wrong network.

On the other hand, suppose the user's prior knowledge is $0 \leftarrow 2$, where 2 must be a parent of 0, and the aim is to find the maximum likelihood network where that is true. If we consider $W = \{0,2,3\}$, where the sink is 0 and 2 is an element of the set u is considered $(i = 0 \,\&\, 2 \in u)$. Although this will always return a network satisfying the respective constraint, it cannot guarantee that it will produce a network with a high score, as the best sink could be $3(3 \leftarrow 0 \leftarrow 2)$. As a result, it is not necessary to consider whether there is a particular arrow or not when we are choosing the best sink. However, we have not examined the case where $2 \leftarrow 0$ (i=2 & u={0,3}), and the rest of the scores need consideration ((i=0 & u={2,3}) and (i=3 & u={0,2})). A constraint is needed to help obtain a network that meets this prior knowledge:

1. Suppose user prior knowledge is $0 \leftarrow 2$ and $\{0,2\} \subseteq W$

   $Then, 2\ not\ the\ sink\ when\ 0 \in U$

2. $if\ (i = 0\ \&\&\ 2 \in U)$

   Then

   BLS(i, U) only returns the parent set that contains 2.

Therefore, from the dynamic programming algorithm, step 2: finding the best sinks

$\text{Prior}_\text{i} = 0$
$\text{Prior}_\text{U} = 2$

- For every $W \subseteq V$ in order Do
  - $Scores[W] \leftarrow 0.0$
  - $sinks[W] \leftarrow -1$
- Loop for all $i \in W$ Do
  - $U \leftarrow W \setminus \{i\}$
  - $BLS(i, U)$ +only returns the parent set that contains .Prior_U
  - $score \leftarrow BLS(i, U) + scores[U]$
    - $if\ (\ sinks[W] = -1\ or\ skore > scores[W]\ )$

      &&
      $if(\ NOT\ ((i == \text{Prior}_\text{U})\ \&\&(\ \text{Prior}_\text{i} \in U)))$
      Do
      - $Scores[W] \leftarrow score$
      - $Sinks[W] \leftarrow i$

5.2.3 Known ordering

The dynamic programming algorithm attempts to find the best ordering and after it finds it, the rest is easy. If we knew the ordering then most of the algorithm would not happen. For example, suppose a Bayesian network and the variables are [0,1,2,3]. If we know that [1,3,2,0] is the right ordering, then it is easy.

However, suppose we only knew that 0 has to come before 2 (0 < 2), and did not know anything else. The aim is to check whether it is possible for the exact dynamic programming algorithm to find the best network so that an ordering respects 0 < 2.

❖ ***0<2 (2 cannot come earlier 0)***

Suppose user's prior knowledge is 0 < 2 and $\{0,2\} \subseteq W$

$Then, 0\ not\ the\ sinks\ when\ 2 \in U.$

Therefore, from the dynamic programming algorithm, step 2: finding the best sinks

$\text{Prior}_\text{i} = 0$
$\text{Prior}_\text{U} = 2$

- For every $W \subseteq V$ in order Do
  - $Scores[W] \leftarrow 0.0$
  - $sinks[W] \leftarrow -1$
- Loop for all $i \in W$ Do
  - $U \leftarrow W \setminus \{i\}$
  - $BLS(i, U)$ + only return the parent set that contains Prior_U
  - $skore \leftarrow BLS(i, U) + scores[U]$
    - $if\ (\ sinks[W] = -1\ or\ skore > scores[W]\ )$

      &&

      $if\ NOT((i == \text{Prior}_\text{i})\ \&\&\ (\text{Prior}_\text{U} \in U))\ Do$
      - $Scores[W] \leftarrow skore$
      - $Sinks[W] \leftarrow i$

5.3 Hill Climbing with Prior Knowledge Algorithm (HCPK)

This research describes how different sorts of prior knowledge are incorporated into the developed learning algorithms. There are many types of prior knowledge, including the knowledge of whether or not node A is a parent of node B and known topological ordering. The most challenging type of prior knowledge, and the main subject of this research, is known ancestor relations and conditional independence. However, the main issue is that more complicated prior knowledge cannot be incorporated into local score. In addition to this, once we have complicated prior knowledge, simply using hill climbing without changing it will fail because it will constantly generate networks that are not allowed. Therefore, we need to add some intelligence.

5.3.1 Arrows which must be absent $A \nleftarrow B$

The user can specify $A \nleftarrow B$ to represent the prior knowledge that B must not be a parent of A. This hard constraint can easily be incorporated as prior knowledge by eliminating all violating parent sets. Therefore, this rule out situations where the parent set does not fit with this prior knowledge.

5.3.2 Arrows which have to be there $A \leftarrow B$

With this prior knowledge, an arrow can be added to a particular child based on the given parents. A child is selected first, and then a decision is made in terms of which parents to add. The HCPK algorithm allows the network for many possible parents to be learnt. For example, take $A \leftarrow B$, where B must be a parent of A. Assume that A is the child specified by the user and that B is the parent specified by the user. In Algorithm 1 line 11, a constraint is needed to help obtain a network that meets this prior knowledge.

- Users can specify either entire parent sets or just part of a parent set. If users specify part of a parent set, then if the score is high, more possible parents are added. But, if users specify the entire parent sets, then only the specified parents are considered.
- The algorithm needs to ensure that A cannot be the ancestor of B.

Figure 12 and Figure 13 show the results of applying HCPK with random restarts to different learning problems for synthetic data generated by the insurance network. The score of the best network found so far was plotted against the number of restarts. In Figure 12, we see that adding prior knowledge leads to a better result. However, when we have more prior knowledge (as shown in Figure 13) the optimal network is found at 159. We used GOBNILP to find the optimal networks, and all prior knowledge is consistent with the optimal network (Cussens, 2011).

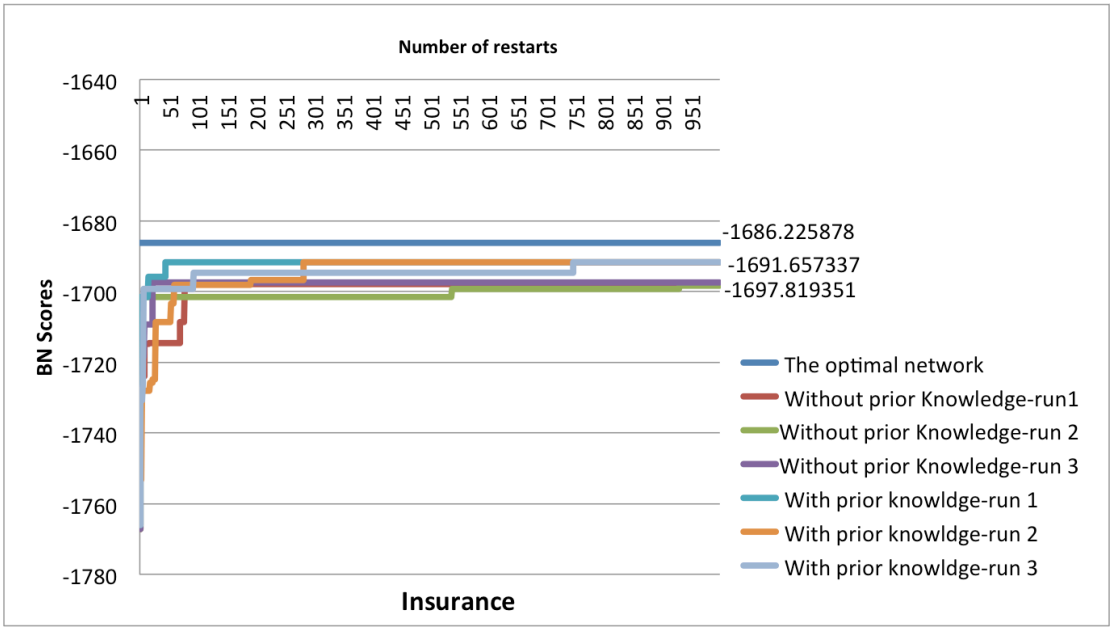

Figure 12: Arrows which have to be there.

The figure shows some runs without prior knowledge and the result of incorporating user's prior knowledge. Here the user specified the parent set $\mathbf{23} \leftarrow \mathbf{11}, \mathbf{22}$ or each of the1,000 restarts.

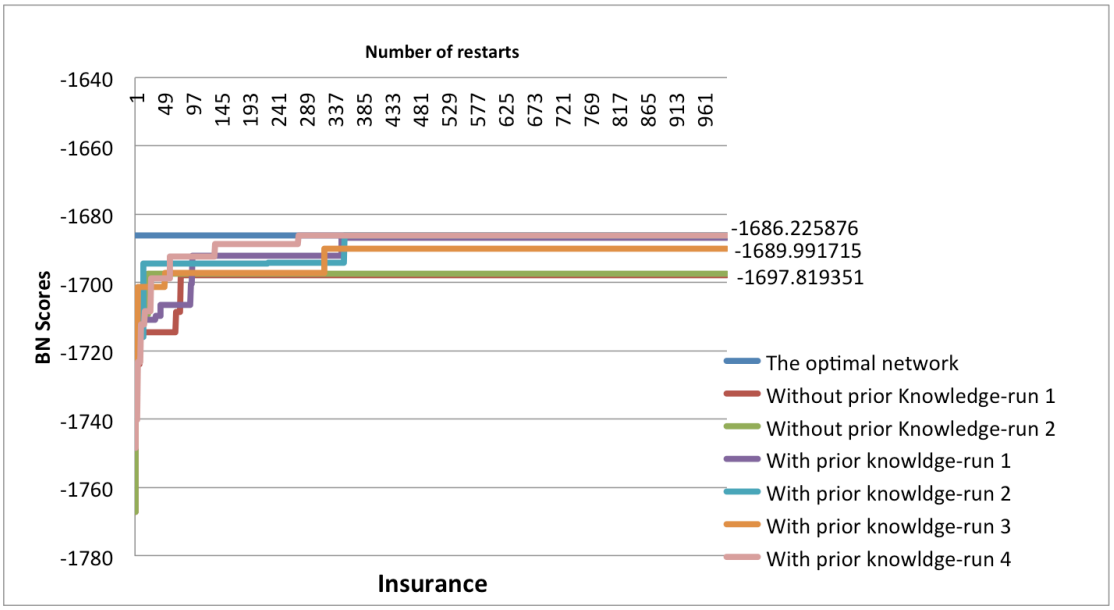

Figure 13: More prior knowledge.

User prior knowledge $(\mathbf{17} \leftarrow \mathbf{7}, \mathbf{16}); (\mathbf{24} \leftarrow \mathbf{15}); (\mathbf{25} \leftarrow \mathbf{14}, \mathbf{20}); (\mathbf{26} \leftarrow \mathbf{3}, \mathbf{21}, \mathbf{24})$ for 1,000 restarts. More Prior knowledge has more effect in the learning algorithm, which sometimes get to the optimal.

5.3.3 Ancestor Relation

Sometimes, a network that does not meet a user's prior knowledge is generated by an algorithm. This network does not present a problem if the user's prior knowledge is simple, like deleting or adding edges. Algorithm 1 is used here to obtain a network that satisfies the constraint. However, if the prior knowledge deals with an ancestor relation, the impossibility of creating a legal network can go undetected until extremely late in the process, which creates problems. The algorithm works by considering a number of possibilities at various points. The algorithm does not consider every possible network; instead, it makes several choices and sticks with them. Overall, it is not hard to check whether a particular ancestor relation is there. However, it is difficult to ensure that the graph we are building will satisfy a given ancestor relation.

5.3.3.1 Backtrack Approach

The primary challenge is to decide what action to take when a branch of the search fails or reaches a dead end. The backtrack approach overcomes this issue by returning to an earlier point that might fix the problem (Russell & Norvig, 2010). Therefore, if the generated network does not meet the user's prior knowledge, Algorithm 3 will perform backtracking. If the generated network is inconsistent with the user's prior knowledge, then Algorithm 4 backtracks to a node in the search tree. Only ancestor relations are considered when the algorithm selects a random variable to backtrack. This random variable is selected from the ancestor of the child specified by the user, and then a new parent set is chosen. The algorithm keeps track of the parent sets that have previously been selected with respect to the backtrack point in order to avoid selecting them again and reducing the structure space. However, a limited portion of the most selected parent sets is retained. When the algorithm reaches the limited number of the most selected parent set, it starts the search again with a new random ordering in order to avoid an infinite loop.

If no legal value for parent sets can be found for the child variable, this will produce an illegal network structure. As a result, the algorithm backtracks to the beginning of the network structure: the first child variable for the particular ordering V.

5.3.3.2 Restarts And Backtrack Approach

In this method, we employ a greedy search until we hit a local maximum. Then, we randomly change the ordering of the variable of the network structure and repeat the process for a certain number of iterations. In addition to this, using several random restarts may be a better strategy if the number of nodes is large and the optimisation is more likely to be complex. In contrast, backtracking is often suitable when a network structure is inconsistent with a constraint. One of the goals of backtracking is to backtrack to the most recent point that might solve the problem and then attempt to find different values. For any single random restart, a backtrack method is applied if the produced network is inconsistent with a user's prior knowledge. Figure 14 shows the results of applying HCPK with random restarts to different learning problems when we plot the score of the best network found so far against the number of restarts. For example, take the prior knowledge $B \dashleftarrow A$ which indicates that A must be an ancestor of B. Assume that B is the specified child by the user and that A is the specified ancestor by the user. If there is an ancestor relation algorithm 3 checks that it is satisfied by an extra check at line 12. These constraints are needed to generate a network that meets this prior knowledge.

- Variable B cannot have an empty parent set
- The algorithm needs to ensure that B cannot be ancestor of A

If these constraints are true, we denote this by:

$$constraint(A, B) = True$$

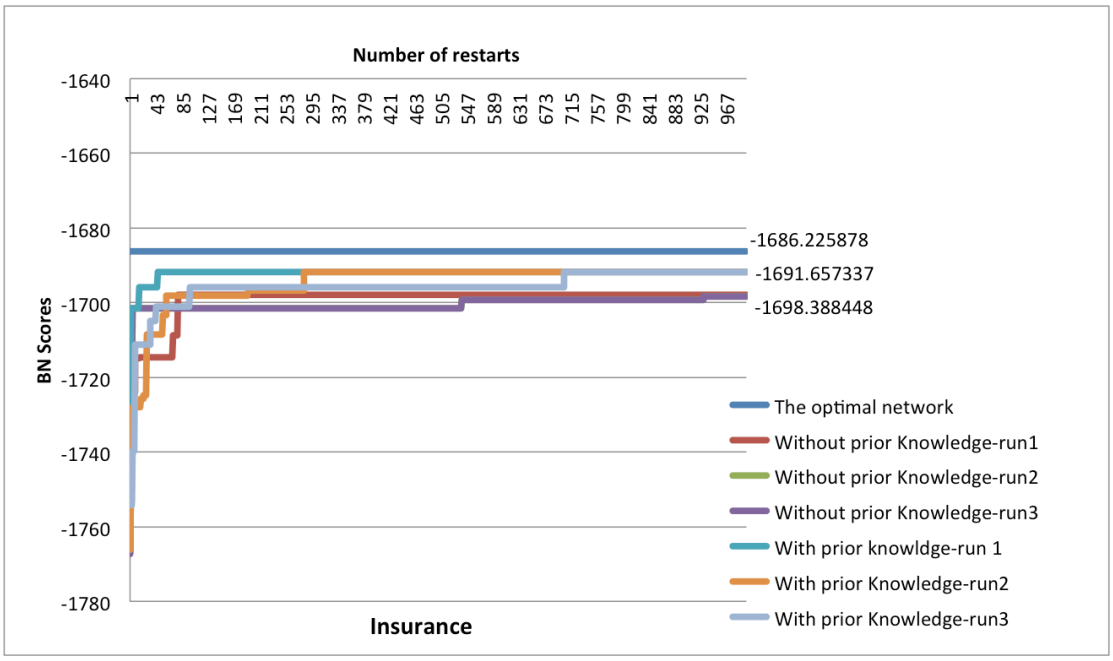

Figure 14: Ancestor relation prior knowledge.

The figure shows some runs without and some with user prior knowledge **6** ⇠23, which indicates that 23 must be an ancestor of 6. It shows the results of applying HCPK with random restarts to different learning problems for synthetic data generated by the insurance network. The score of the best network found so far was plotted against the number of restarts.

Algorithm 3: Restart and backtrack.

```
        REQUIRE: RemovingList {Contains all the children with most visited parents }
        REQUIRE {Parameter number to limited number of the most visited parent sets}
        REQUIRE g {a graph}
1.          For i = 0 to length(V) Do
2.                  BP_a(x) ← [ ]  // Best Parents Combination
3.                  x ← V[i] // child
4.                   if  not RemovingList. contains(x) then
5.                       continue
6.                  end if
7.                  PP_a(x) ← V − x // Possible parents
8.                  bestLocalScore ← Score(x, BP_a(x))
9.                  For  j = 1 to length(PP_a(x))  Do
10.                     P_a(x) ← possibleParents[j]
11.                     BP_a(x). add[P_a(x)]
12.                     IF CheckThatParentIsOk(P_a(x)) == FALSE then
13.                             Continue
14.                     END IF
15.                     localScore  ← Score(x, BP_a(x)) {Using BDe score}
16.                     IF bestLocalScore  <
                        localScore AND checkCyclicMap(g , x, BP_a(x))AND(constraint  apply)
                        AND disjoint(RemovingList. get(x), BP_a(x))
17.                     Then
18.                             bestLocalScore  ← localScore
19.                              g. update(x, BP_a(x))
20.                     ELSE
21.                             BP_a(x). Remove[P_a(x)]
22.                     END IF
23.                     list  ←  RemovingList. get(x)
24.                     if {list  == null or list. size()  >  P)
25.                     then
26.                      RemovingList. put(x, BP_a(x))
27.                     else
28.                             list. addAll(BP_a(x))
29.                             RemovingList. put(x, list)
30.                     end if
31.                 end for
32.         end for
33.     if  illegal(g) then
34.        ancestorRelation/conditional independence(V, true)
35.      else
36.        ancestorRelation/conditional independence(V, false)
37.     end if
```

Algorithm 4: Ancestor relation

```
     REQUIRE: ordered variables
     REQUIRE: user's prior knowledge about the child
     REQUIRE : fromStart, True or False
1.   position ← 0
2.    removingListMap ← [ ]
3.   if checkAncestorRelationisok(child user's knowledge) == True then
4.          return
5.   end if
7.   IF not formStart then
8.          Backtrack ← ancestor(child). Random
9.   else{ Starting from the beginning}
10.         Backtrack ← allVariables(postion)
11.  end if
12.  SelectedBacktrack ← allVariables. indexOf(Backtrack )
13.  for i ← SelectedBacktrack To allVariables. size() do
14.         removingListMap. add(i)
15.         ancestor. remove(i) {ancestor, contains a list of all the children
             with their ancestors }
16.  end for
17.  Restarts and Backtrack(allVariables, RemovingList)
```

Let us take the prior knowledge that 6 ⇠ 7, which indicates 7 must be an ancestor of 6. Assuming that 6 is the child specified by the user and that 7 is the ancestor specified by the user, then the following constraints are needed to generate a network that meets this prior knowledge:

- Variable 6 cannot have an empty parent set.
- The algorithm must ensure that 6 cannot be an ancestor of 7.

The result of applying HCPK ancestor relations, for synthetic data generated by the Asia 100, is shown in Figure 1:

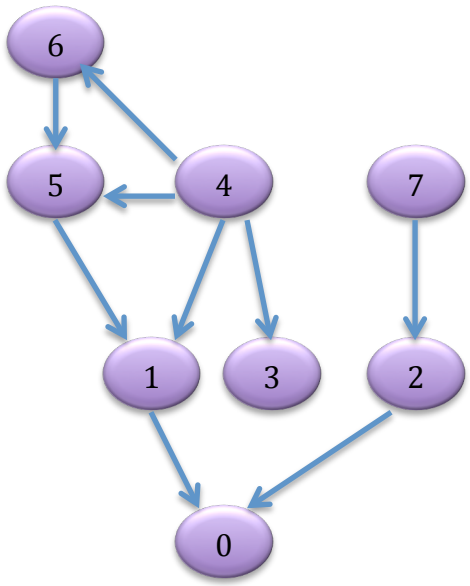

Figure 15: The generated network from HCPK.

The generated network is inconsistent with the user's prior knowledge, and we need to perform backtracking again. The current ancestors of the child specified by the user are [4], as only ancestor relations are considered when the algorithm selects a random variable to backtrack. In this example, Node 4 is selected from the ancestor of the child specified by the user and then a new parent set is chosen. If the score is high, more parents that are possible are added. For Child 4, the parent sets that have previously been selected will not be parents again, as the HCPK algorithm keeps track of the parent sets that have previously been selected with respect to the backtrack point. The generated network is shown in Figure 16.

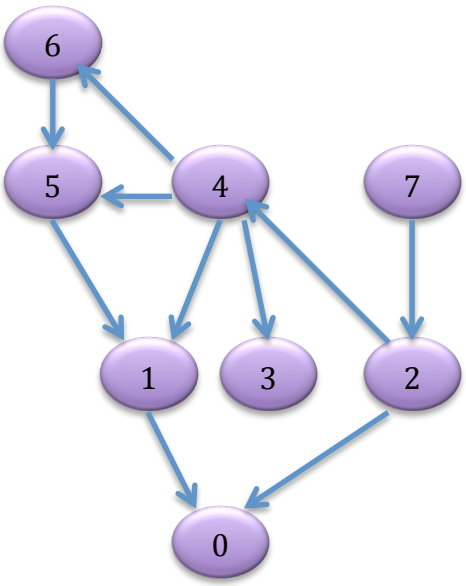

Figure 16: The generated network from HCPK after backtracking.

The generated network is consistent with the user's prior knowledge, and we do not need to perform backtracking. The total score is -248.02. We run several iterations of the HCPK algorithm with different random orderings of the variables to compensate partially for the myopia of hill climbing. Different random orderings result in different scores. Therefore, after a few iterations, the HCPK algorithm found a high scoring network at -245.64.

Generally, when adding prior knowledge, it is consistent with the optimal network, or the true network helps the algorithm to generate a more accurate network. However, it is not necessary the true network have a high scoring network. Adding prior knowledge gives us slightly fewer arrows. Occasionally, the algorithm performs worse without prior knowledge. The HCPK algorithm deals with bigger problems, can incorporate different sorts of prior knowledge, and always returns a network that satisfies user prior knowledge regardless of whether it achieves a high score or not.
Such a randomised restart ensures that the best scoring network that is consistent with the constraints will eventually be produced. However, the use of backtracking (as opposed to a pure random-restart approach) adds intelligence to the search and greatly increases the chances of generating a good network on each iteration.

Overall, when the HCPK algorithm performs backtracking, it keeps track of the parent sets that have previously been selected with respect to the backtrack point to avoid selecting them again, reducing the structure space. Additionally, a limited portion of the most-selected parent sets is retained, and it starts the search again with a new random ordering when it reaches the limit.

Recall that there are two backtrack points (see Section 5.3.3.1 for more details); if the generated network is inconsistent with the user's prior knowledge, then Algorithm 4 backtracks to the following:

- A node in the search tree: Only ancestor relations are considered when the algorithm selects a random variable to backtrack.
- The beginning of the network structure: The first child variable for the particular ordering V if no legal value for parent sets can be found for the child variable.

In addition, the constraints in Section 5.3.3.2 play a big role in the HCPK algorithm to facilitate generating a network that meets the user's prior knowledge and ensures that the graph will satisfy a given ancestor relation. These constraints are needed to generate a network that satisfies a user's prior knowledge and reduces the structure space. Therefore,

(user's Prior knowledge) $\times$(HCPK) $\times$ (Dataset) $\approx$ a network that satisfied user's prior knowledge .

5.3.4 Conditional Independence

Dependencies and independencies are the main issues in a probability distribution. Local independencies in Bayesian networks are where each node is independent of its non-descendants, given its parents. Global independencies are derived from d-separation, which helps to ensure that specific sets of independencies ($A \perp B \mid Z$) hold in a distribution, so that a variable A is conditionally independent of a particular variable B, given its variable Z. In other words, the observation of A changes the belief about B, in the presence of evidence about Z. D-separation is one of the ways of detecting conditional independence relations. In d-separation, Koller and Friedman (2009) demonstrate that there are three main patterns that illustrate whether two variables are independent in the presence of evidence.
In this section, we investigate incorporating the conditional independence prior knowledge into the developed learning algorithm using two different approaches.

5.3.4.1 Restarts and Backtrack Approach

Intelligent backtracking often applies when a network structure is inconsistent with a constraint. For any single random restart, an algorithm checks for conditional independence. For example, take ($A \perp B \mid Z$) as the conditional independence prior knowledge specified by the user, where A is a conditional independent of B given Z. The algorithm uses the d-Separation algorithm to check for conditional independence, and discovers the nodes reachable from A given Z via active trails. If the generated network does not meet the user's prior knowledge of conditional independence, the backtrack method is applied.

In Algorithms 5 and 3, if the current network does not meet the user's prior knowledge then the algorithm backtracks to a node in the search tree. Only conditional independence active trails are considered when the algorithm selects a random variable to backtrack (see Algorithm 5's pseudo-code for details). Meanwhile, if no legal value for parent sets can be found for the child variable, this will generate

an illegal network structure. As a result, the algorithm backtracks to the beginning of the network structure: the first child variable for the particular ordering . In Algorithm 2, a constraint is needed to help obtain a network that meets this prior.

- If Z is current child then $\{A, B\} \not\subset P_a(Z)$

Algorithm 5: Conditional independence

```
      REQUIRE: ordered variables
      REQUIRE: user's prior knowledge about the child
      REQUIRE : fromStart, True or False
1.    position ← 0
2.    removingListMap ← [ ]
3.    R = FindReachable(A, Z)
4.    if { R == True}
5.          return { R is DSeparation algorithm for finding nodes reachable
            from A given Z via active trails }
6.    end if
7.    IF not formStart then
8.          Backtrack ← R(List). Random
            {R is returned list of active trails from D − Separation }
9.    else{ Starting from the beginning}
10.         Backtrack ← allVariables(postion)
11.   end if
12.   SelectedBacktrack ← allVariables. indexOf(Backtrack )
13.   for i ← SelectedBacktrack To allVariables. size() do
14.         removingListMap. add(i)
15.         ancestor. remove(i){ancestor, contains a list of all the children with their ancestors}
16.   end for
17.   Restarts and Backtrack(allVariables, RemovingList)
```

5.3.4.2 Conditional Independence Checks

To investigate the effect of including conditional independence prior knowledge, we used a d-Separation algorithm for each move in the HCPK algorithm. For example, take $(A \perp B \mid Z)$ as the conditional independence prior knowledge specified by the user, where A is a conditional independent of B given Z.

Therefore, for each additional move, the algorithm creates a *temporary graph* that contains the current graph and the possible parent set. It then checks this *temporary graph* for conditional independence. If the possible parent sets do not satisfy the conditional independence checks, it is not considered. Using this approach, we continue to build the graph by conducting these early checks and, eventually, end up with a network that meets the user's prior knowledge. As shown in Figure 15, adding the prior knowledge has a positive effect on the learning. In Algorithm 1, a constraint is needed to help obtain a network that meets this prior knowledge.

- If Z is current child then $\{A, B\} \not\subset P_a(Z)$
- For each additional move, the algorithm checks the current graph for conditional independence; checking that B is not reachable from A given Z via active trails.

  $((A \perp B \mid Z), TemporaryGraph) == True)$

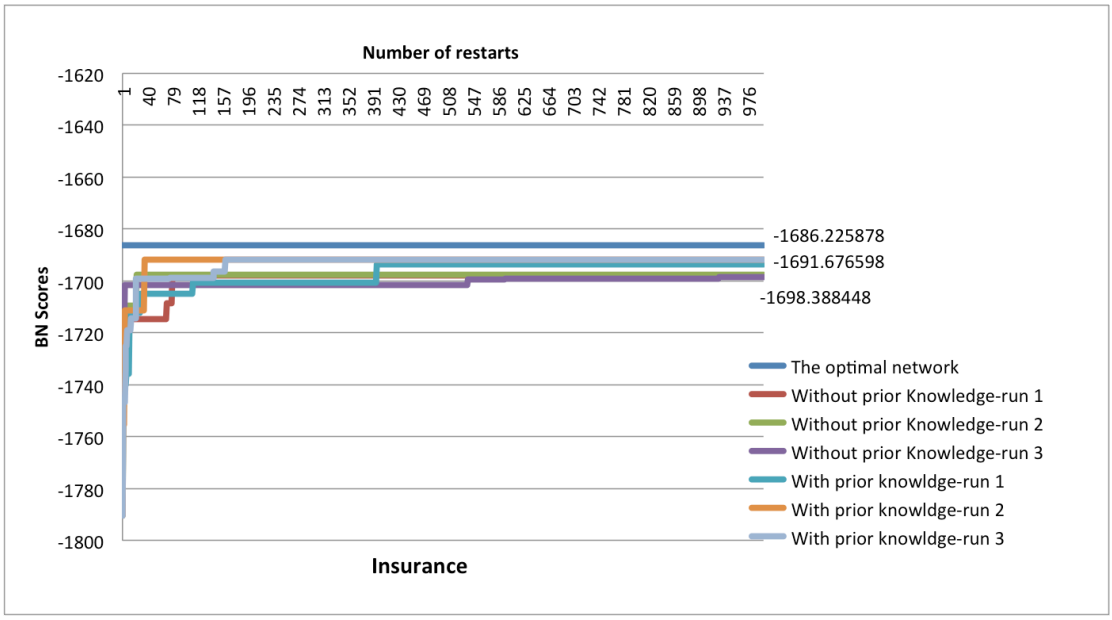

Figure 17: Conditional independence prior knowledge.

The figure shows some runs without prior knowledge and some with user prior knowledge ($\mathbf{11} \perp \mathbf{14} \mid \mathbf{6}$). We plotted the results of the best network found so far against the number of restarts. As shown, adding prior knowledge leads to a better result: the best network was found at 600, and then a better network was found at 617.

In this chapter, we presented an algorithm that can incorporate different types of priors to the developed algorithm. This chapter introduced prior information and highlighted the differences between hard and soft prior information. Then, how different sorts of prior knowledge are incorporated into the developed learning algorithms were described. The results of applying the dynamic programming and HCPK with prior knowledge to different learning problems are given in the following chapter.

6 Results And Evaluation

This chapter presents experiments conducted using a developed algorithm on Dynamic Programming and HCPK, with and without prior knowledge. Here is the main contribution of this thesis: presenting an algorithm that can incorporate different types of prior knowledge to the developed algorithm. Section 6.1 describes the results of applying the dynamic programming with prior knowledge to different learning problems. Section 6.2 investigates whether our current algorithm can result in a high score without prior knowledge and compare it to other existing applications. It then describes how different sorts of prior knowledge are incorporated into the developed learning algorithms (HCPK).

In these experiments, we have used different datasets and a random number of runs. We have used the datasets available at http://www.cs.york.ac.uk/aig/sw/gobnilp/data. In this research, GOBNILP is used to find the optimal networks, and all prior knowledge is consistent with the optimal network.

We have implemented the algorithms described in this thesis and our implementation is written in Java programming language. The experiments to be described next were run under Windows on an ordinary desktop PC with a 2.4GHz Pentium processor and 2.0GB of memory.

6.1 Dynamic Programming Algorithm

Evaluation of the accuracy of a structure-learning algorithm for this experiment is based on a comparison between the output produced by the developed algorithm of the dynamic programming and GOBNILP. In this research, GOBNILP is used to find the optimal networks, and all prior knowledge is consistent with the optimal network. The next section include full details of the result of incorporating different types of prior knowledge to the dynamic programming algorithm.

In reference to efficiency, the dynamic programming algorithm was efficient and the speed was acceptable because the local scores were sorted in a decreasing order. In addition, with prior knowledge, one can often speed up the algorithm. When we do the main loop to find the best sinks for sets, if we have prior knowledge, it speeds up the algorithm a bit. However, the step involved with finding the best sinks has the greatest computational complexity: $BLS(i, U)$ has complexity $O(n^2)$. In this algorithm, the for loop for a given $W \subset V$ is called $1 \leq |W| \leq n$ times and for each of the 2^n subsets of V. Moreover, each for loop call also has complexity $O(n^3)$. Therefore, the computational complexity of the algorithm is, at worst, $O(n^3 2^n)$.

The dynamic programming algorithm is a good one because if it terminates, it will return the optimal network. Moreover, if users wish, they can input prior knowledge; if they do not have prior knowledge, the algorithm still works. However, the problem is that the algorithm has certain limitations by itself. The algorithm is limited by the amount of memory, which has to be used. This algorithm is an attempt to search for a pedigree of up to approximately 31 individuals. The dynamic programming algorithm approach will fail if we have too many variables because it will just run out of memory. Another problem is that, in the dynamic programming algorithm, we have to compute all of the local scores first, but, if we have many variables and we allow variable possibilities in terms of having too many parents, then computing the score does not work because there are too many of them.

Here, we show the results of incorporating prior knowledge to the dynamic programming to different learning problems. In these experiments, we have used different datasets. The results of applying the dynamic programming to different learning problems for synthetic data generated by the Asia (8 variables) and Kredit (18 variables) networks. For variables more than 30, the execution has stopped because the program has run out of available memory. The score found by the developed dynamic programming in Figures 16 to 25.

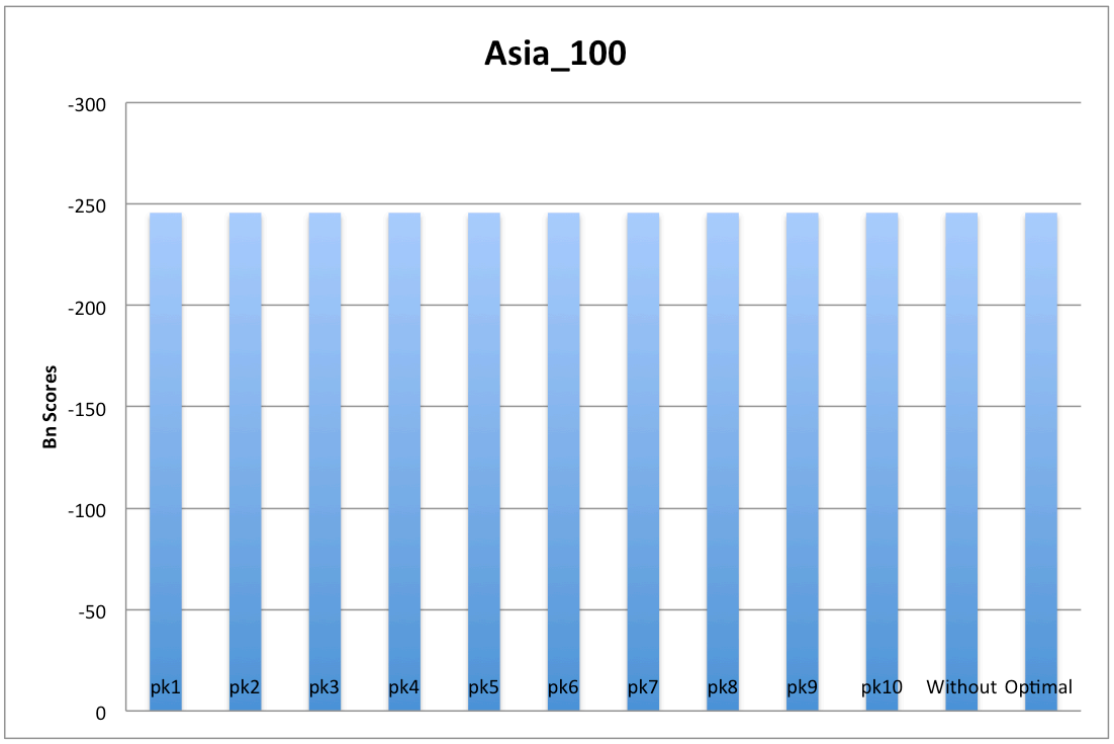

Figure 18: The results of applying the dynamic programming with prior knowledge consistent with GOBNILP, arrows which have to be there and known ordering, for synthetic data generated by the Asia 100.

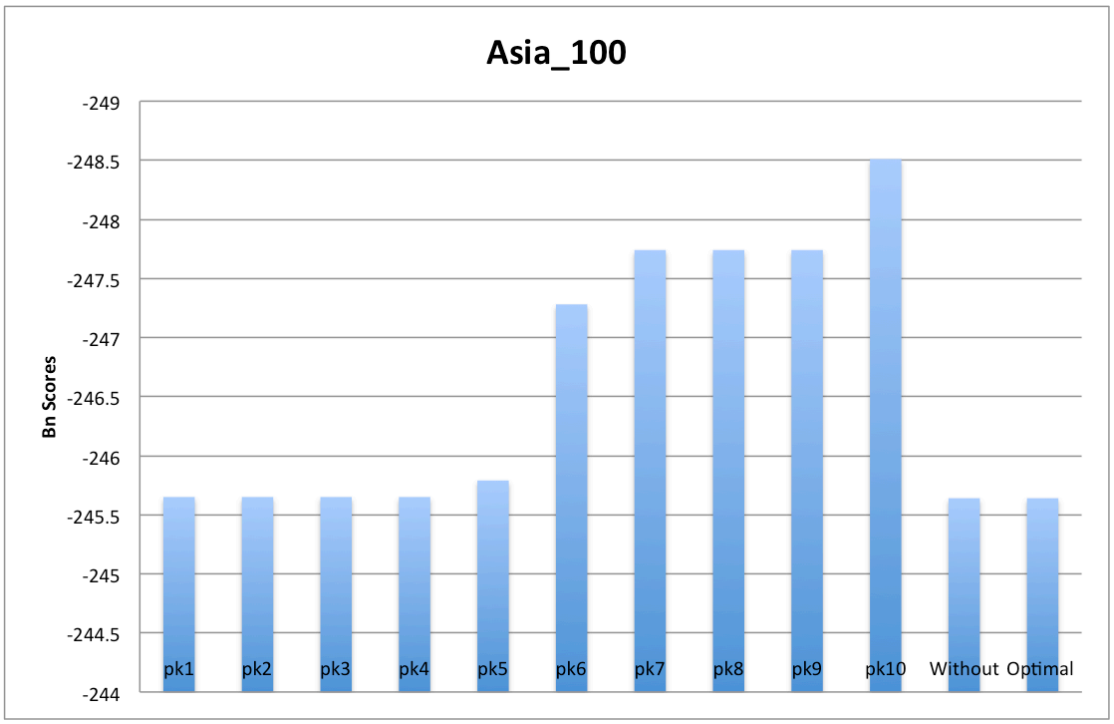

Figure 19: The results of applying the dynamic programming with prior knowledge inconsistent with GOBNILP, arrows which have to be there, for synthetic data generated by the Asia 100.

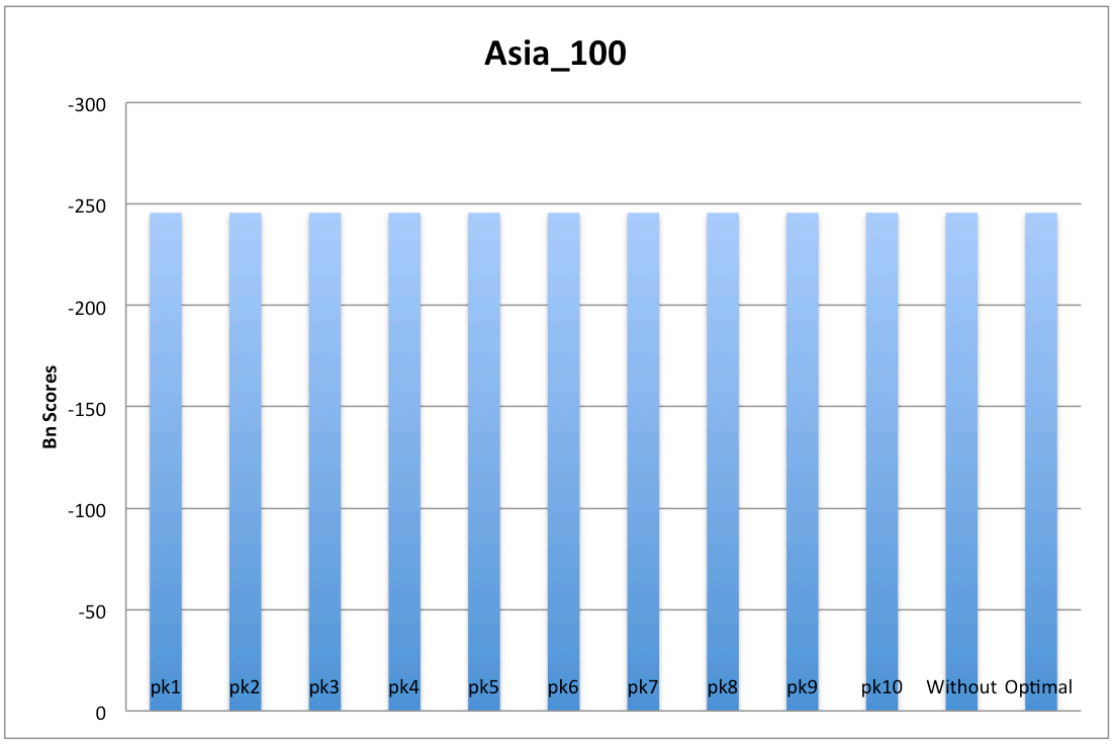

Figure 20: The results of applying the dynamic programming with prior knowledge inconsistent with GOBNILP, known ordering, for synthetic data generated by the Asia 100.

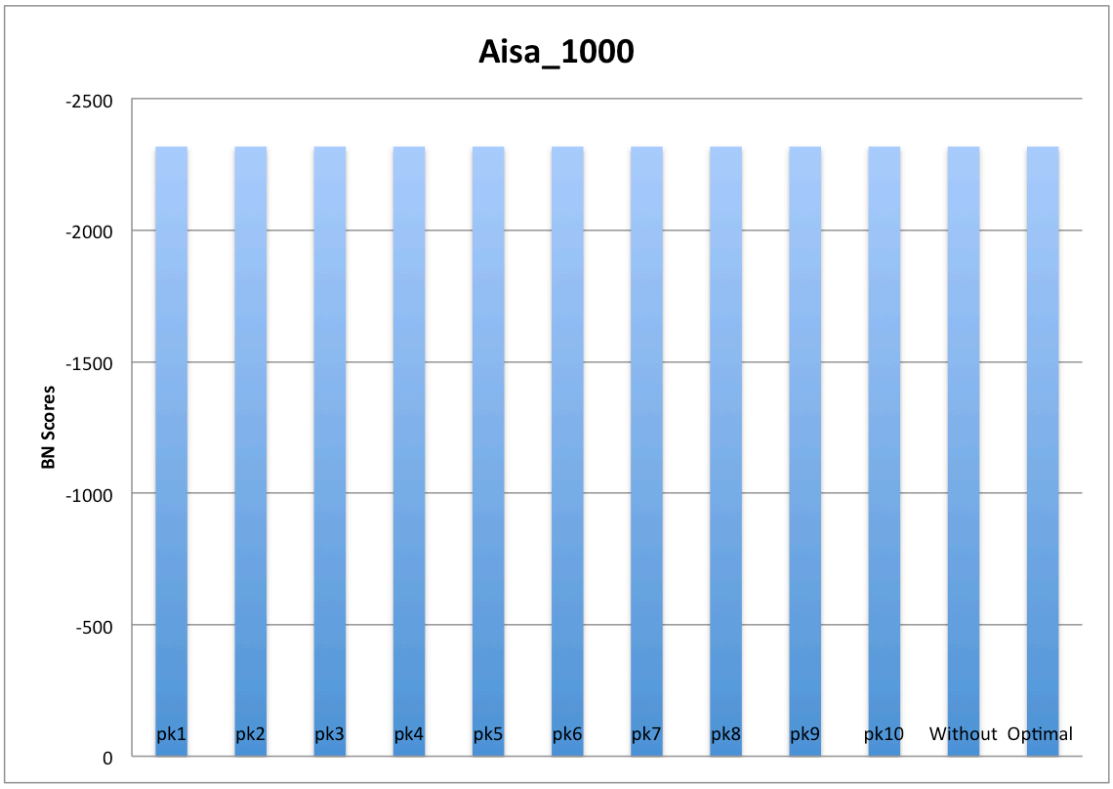

Figure 21: The results of applying the dynamic programming with prior knowledge consistent with GOBNILP, arrows which have to be there and known ordering, for synthetic data generated by the Asia 1000.

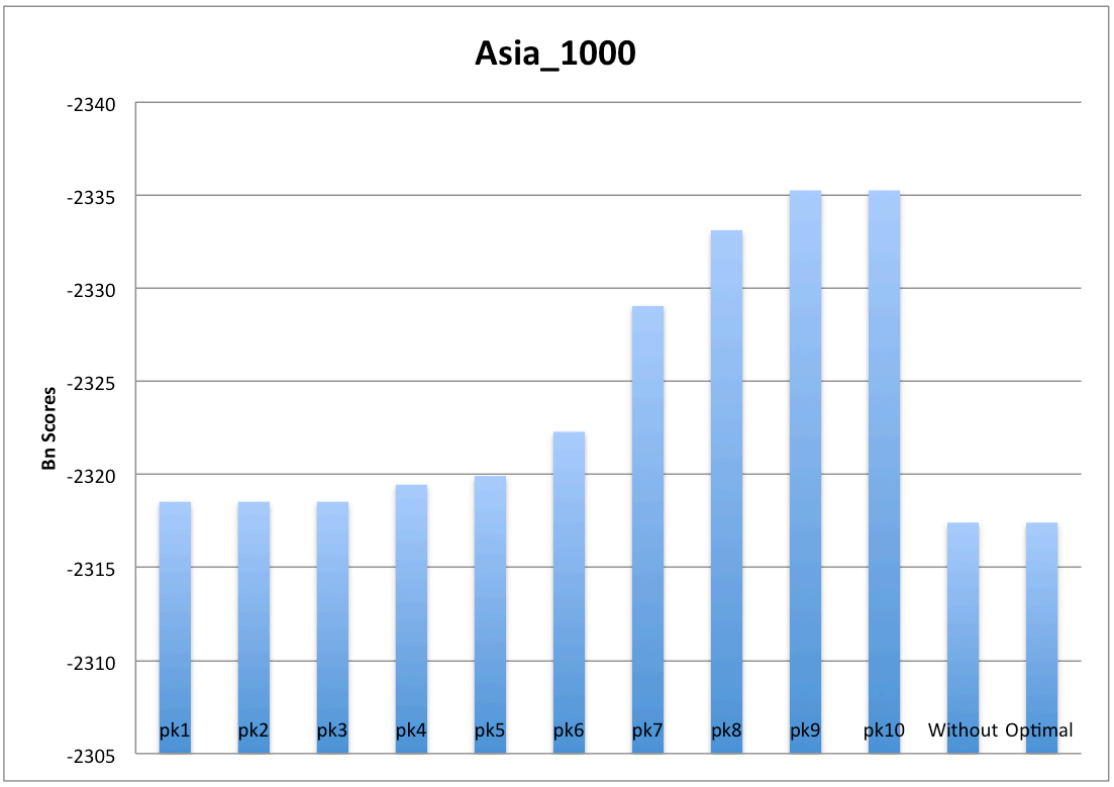

Figure 22: The results of applying the dynamic programming with prior knowledge inconsistent with GOBNILP, arrows which have to be there, for synthetic data generated by the Asia 1000.

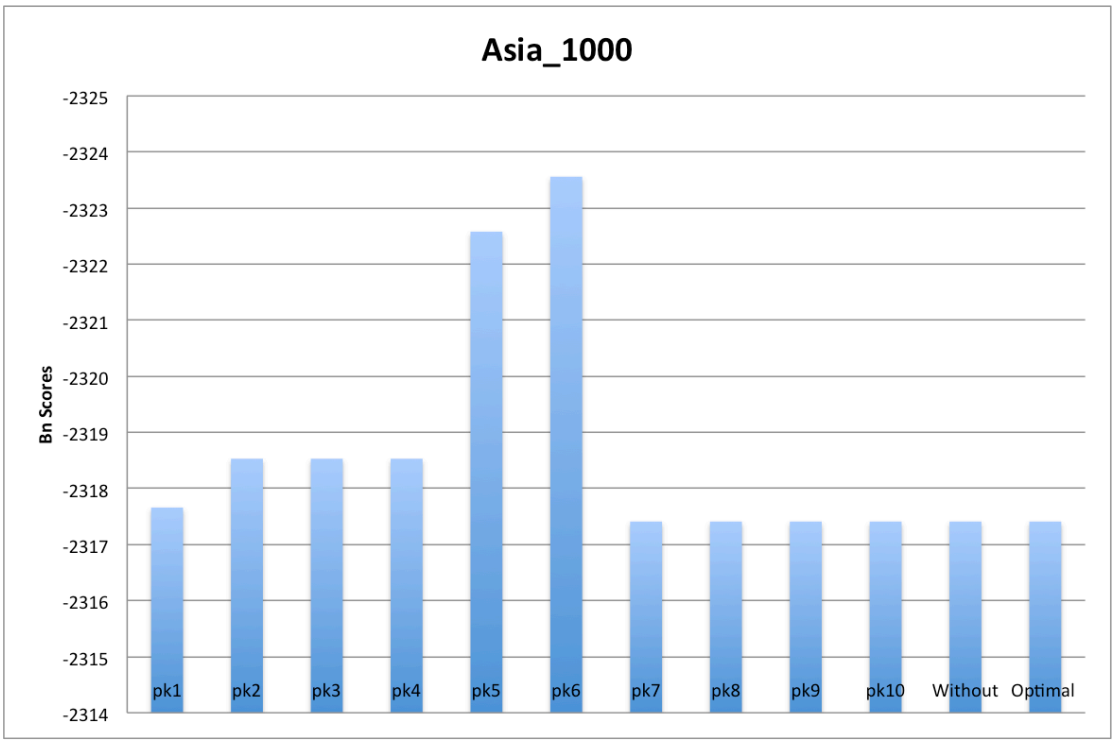

Figure 23: The results of applying the dynamic programming with prior knowledge inconsistent with GOBNILP, known ordering, for synthetic data generated by the Asia 1000.

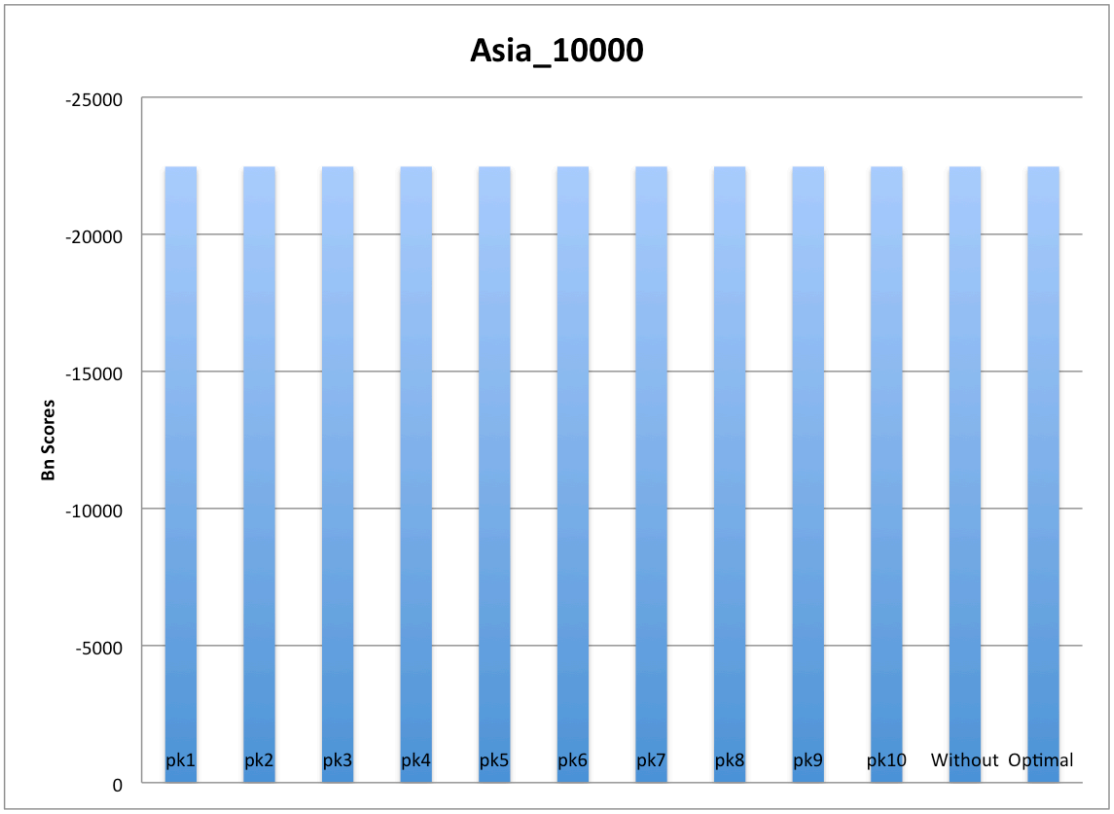

Figure 24: The results of applying the dynamic programming with prior knowledge consistent with GOBNILP, arrows which have to be there and known ordering, for synthetic data generated by the Asia 10000.

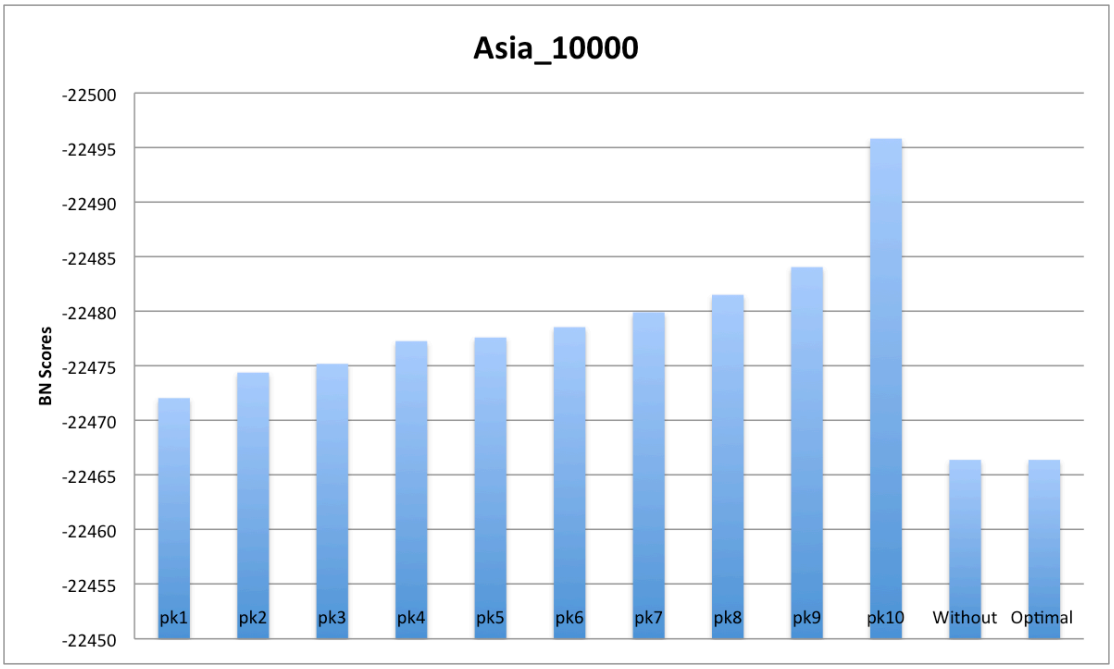

Figure 25: The results of applying the dynamic programming with prior knowledge inconsistent with GOBNILP, arrows which have to be there, for synthetic data generated by the Asia 10000.

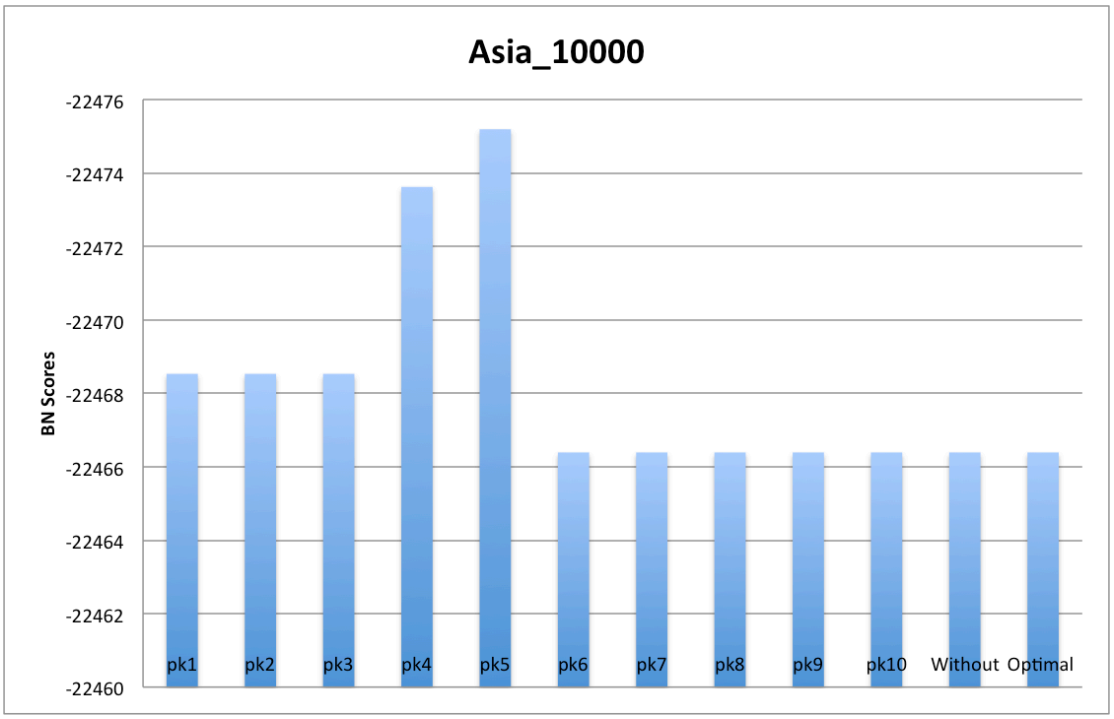

Figure 26: The results of applying the dynamic programming with prior knowledge inconsistent with GOBNILP, known ordering, for synthetic data generated by the Asia 10000.

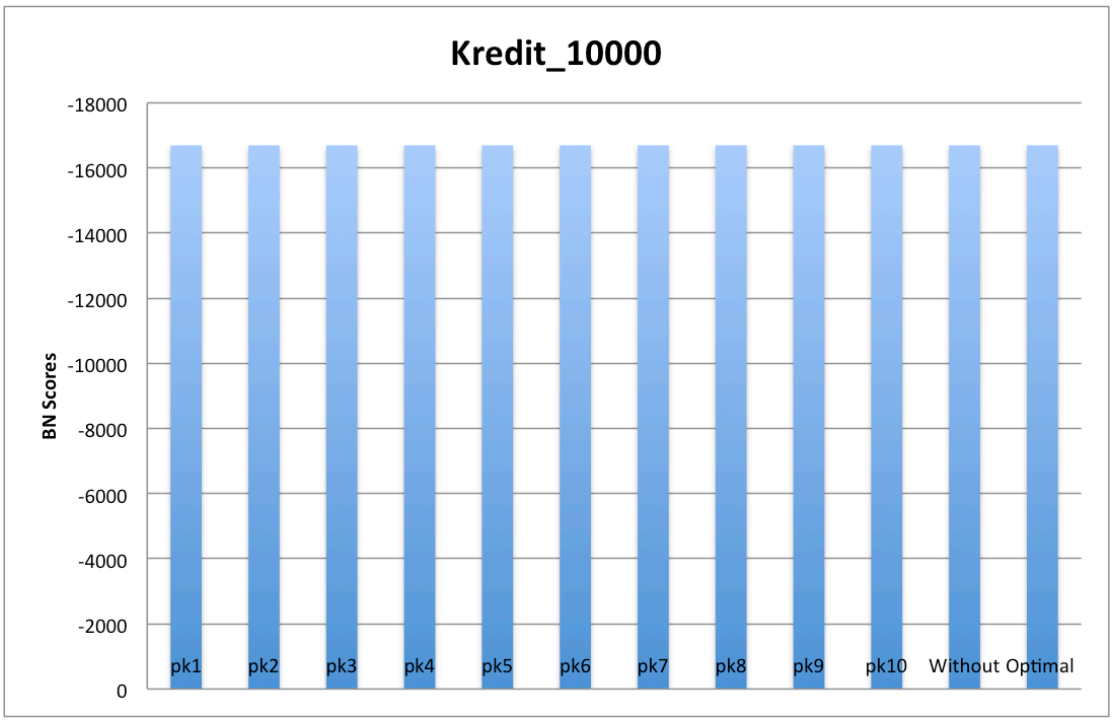

Figure 27: The results of applying the dynamic programming with prior knowledge consistent with GOBNILP, arrows which have to be there and known ordering, for synthetic data generated by the Kredit 10000.

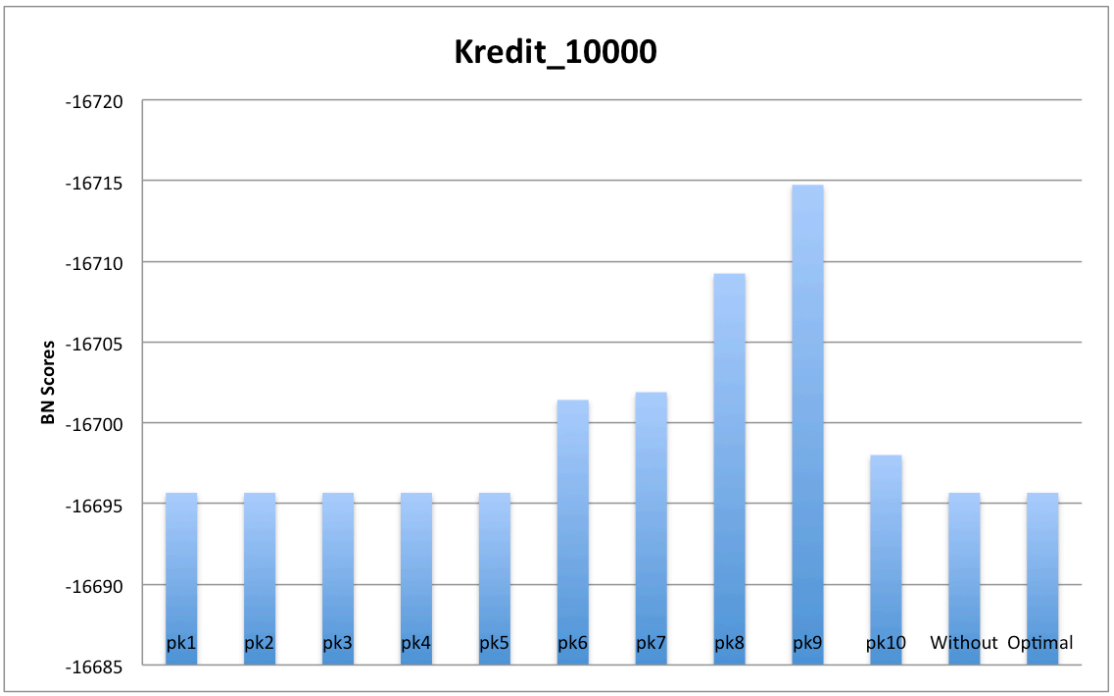

Figure 28: The results of applying the dynamic programming with prior knowledge inconsistent with GOBNILP, arrows which have to be there and known ordering, for synthetic data generated by the Kredit 10000.

The results of applying the dynamic programming with prior knowledge to different learning problems for synthetic data were generated by the Asia, Kredit, insurance, Alarm, Carpo, and Diabetes networks. The score of the best network found so far against the number of restarts as shown in Figures 16, 19, 22. It shows the results of incorporating prior knowledge arrows, which have to be there. With this prior knowledge, an arrow can be added to a particular child based on the given parents.
For datasets, aisa100, asia1000, and aisa10000 (8 variables), it always found the optimal network (Please see details in Appendix-A).

Figure 25 shows the results for the dataset Kredit 10000 (18 variables) wherein the optimal network was also always found. It also shows the results of incorporating prior knowledge arrows, which have to be there, as well as knowing ordering.
In dynamic programming, prior knowledge consistent with GOBNILP always results in the optimal network. Inconsistent prior knowledge has quite a negative effect on the learning. As the result shows that if we incorporate inconsistent prior knowledge, we get a consistently worse score as shown in Figures 17,18, 20, 21 23, 24, and 26.
However, if we have inconsistent prior knowledge, sometimes we got a worse network. There's a Bayesian network, which is a Markov equivalent in which the GOBNILP found to be consistent with this order. For most networks, there are several orderings we can have, and most networks have some Markov equivalent wherein they have different ordering. Thus, it is not a big surprise that we can change the constraints in the ordering and still get the same score. But, if we specify an ordering or parent set that is not in a v-structure, then this will not have beneficial effect on learning. For example, from GOBNILP 10→17←16 was a v-structure in a Kredit dataset, and if we specify a parent set of a variable where there was not a v-structure, then we get a low score. For example, pk9 when a user specifies an ordering 17<10, which is 17 comes earlier than 10, then we have a worse score, as it shown in figure 26.
Nevertheless, variables more than 30 (insurance, Alarm, Carpo, and Diabetes), the execution has stopped because the program has run out of available memory.

6.2 HCPK

In this section, we show the results of applying HCPK with random restarts to different learning problems for synthetic data. In these experiments, we have used different datasets and a random number of runs. We have used the datasets available at http://www.cs.york.ac.uk/aig/sw/gobnilp/data. In this research, GOBNILP is used to find the optimal networks, and all prior knowledge is consistent with the optimal network. In these experiments, Bayesian networks score are averages. The averages were taken over four different runs.

The results of applying HCPK with random restarts to different learning problems results in synthetic data generated by the Asia, Kredit, insurance, Alarm, Carpo networks, as well as the score of the best network found so far against the number of restarts. For datasets, Asia 100, Asia 1000, and Asia 10000 (8 variables), it always found the optimal network. Also, for the dataset Kredit 10000 (18 variables), the optimal network was always found.

To investigate whether our current algorithm can get a high score without prior knowledge and compare it to other existing application, we ran three freely available programs on nine different datasets. Table 1 shows the comparison between different applications, where '1' is the HCPK algorithm restricted to a maximum of three parents per node, '2' is the HCPK algorithm with no restriction on the number of parents, 'B8a' is Banjo (Greedy) restricted to a maximum of eight parents, 'B3a' is Banjo (Greedy) restricted to a maximum of three parents, 'B8s' is Banjo (SimAnneal) restricted to a maximum of eight parents, and ' B3s ' is Banjo (SimAnneal) restricted to a maximum of three parents. Also, GOBNILP was restricted to a maximum of three parents. However, - indicates the maximum number of states that a variable can assume, limited to 7, and * indicates that execution has stopped because the program has run out of available memory.

For many cases, simple HCPK gets quite close to the optimal network. Table 1 shows that the current algorithm *without* prior knowledge is generally slightly worse than

Banjo. Also, although GOBNILP can find optimal networks on these small examples, it will have problems due to the fact that, for example, nodes can have many parents. However, our current algorithm does not have this problem and can incorporate different sorts of prior knowledge and more complicated knowledge such as ancestor information and conditional independence, which cannot be incorporated into local scores, thereby leading to a much better network. For bigger datasets, including a sample size of 10000 generated from this network, which contains more then 400 discrete variables (with 10 to 20 levels each) it was hard to use Bnlearn as it requires the levels for each variable to be written manually.

Table 1: Comparison software

	Gobnilp	1	2	B8a	B3a	B8s	B3s	Bnlearn
Alarm_37	-1345.5	-1380.1	-1376.2	-1370.9	-1378.1	-1344.4	-1348.2	-1431.7
Insurance-27	-1667.9	-1678.2	-1677.1	-1679.1	-1678.6	-1670.7	-1674.9	-1730.4
Mildew_35	-5968.3	-6420.1	-6420.1	-	-	-	-	-6532.8
Asia_8	-243.6	-243.6	-243.6	-243.6	-243.6	-243.6	-243.6	-248.97
Carpo_60	-1825.7	-1856.0	-1844.6	-1971.3	-1975.7	-1851.1	-1858.5	-1933.1
Hailfinder-56	-6019.4	-6021.4	-6019.8	-	-	-	-	-6222.5
kredit	-16694.3	-16694.3	-16694.3	-	-	-	-	-16804.2
Pigs-441	*	-41980.9	-42003.4	*	*	*	*	-43112.5
Diabetes-413	*	-59695.4	-56230.1	-	-	-	-	

Restarts averaging

In this section, we show the results of applying HC without incorporating prior knowledge, with random restarts to different learning problems for synthetic data.

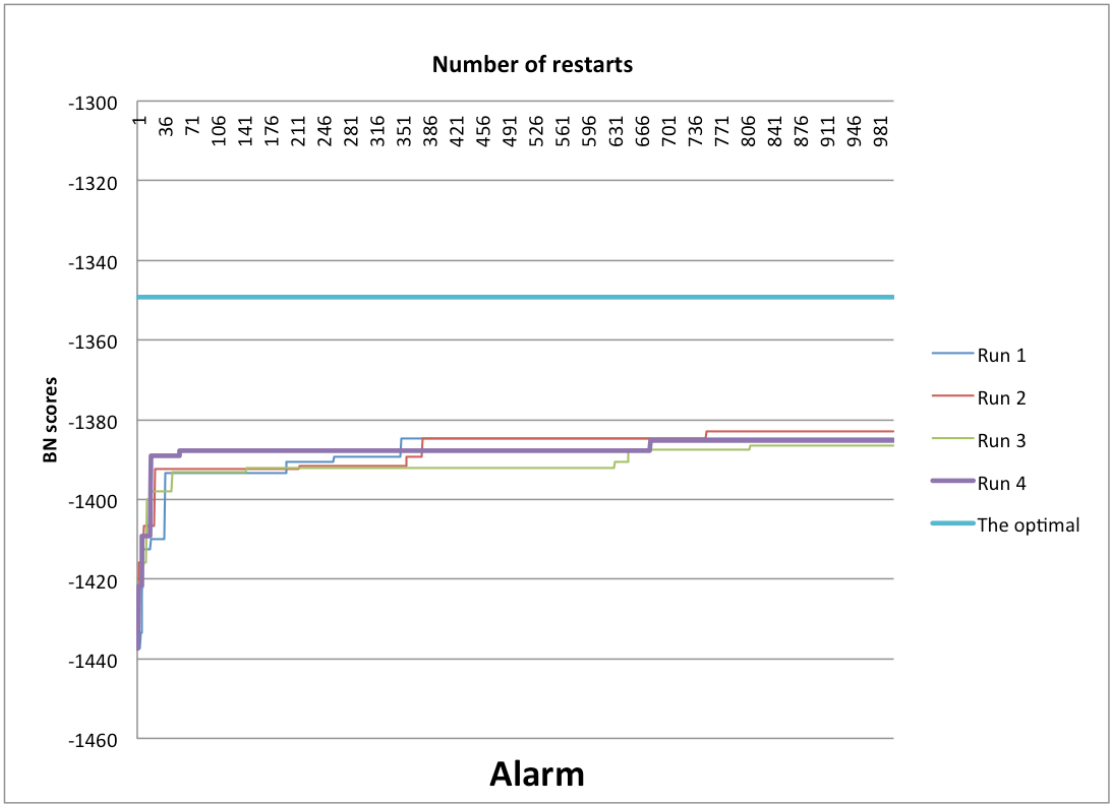

Figure 29: Alarm network without prior knowledge

Figure 27 shows the results of applying HC without prior knowledge with random restarts for synthetic data generated by the alarm network when we plot the score of the best network found so far against the number of restarts. Each run has 1,000 random restarts.

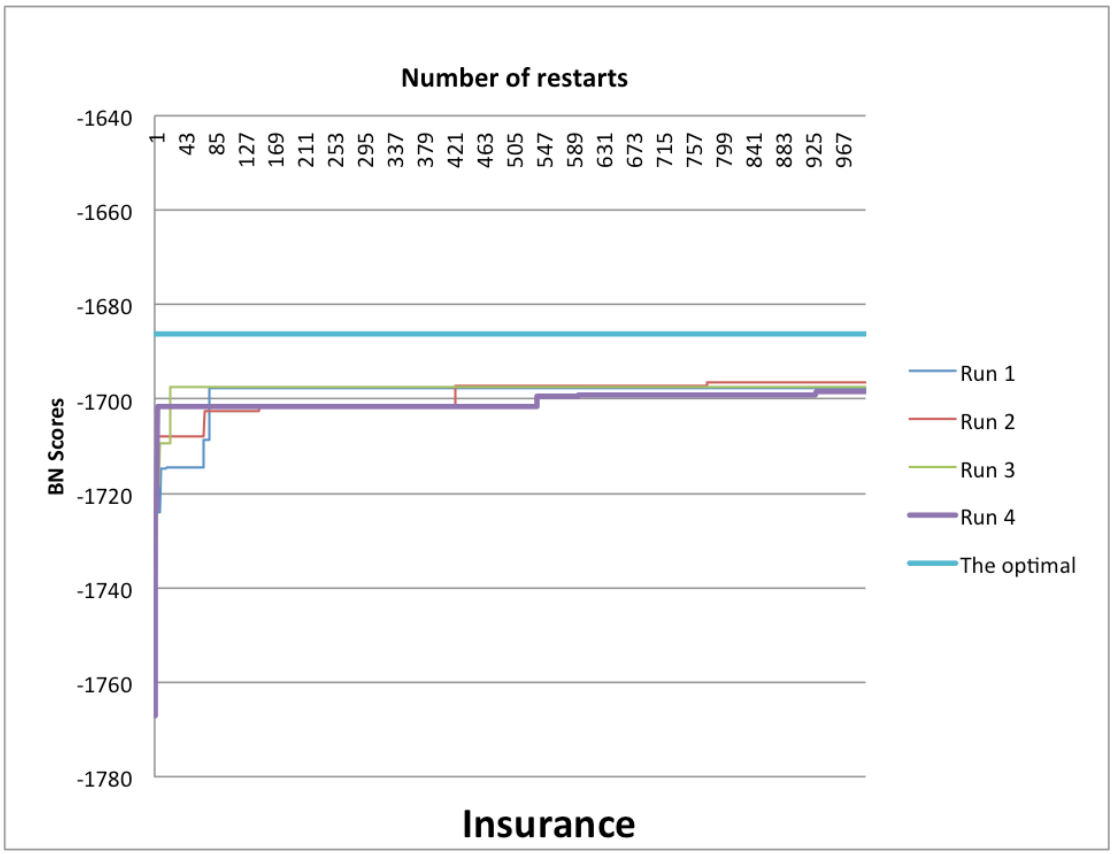

Figure 30: Insurance network without prior knowledge.

Figure 28 shows the results of applying HC without prior knowledge with random restarts for synthetic data generated by the Insurance network when we plot the score of the best network found so far against the number of restarts. Each run has 1,000 random restarts.

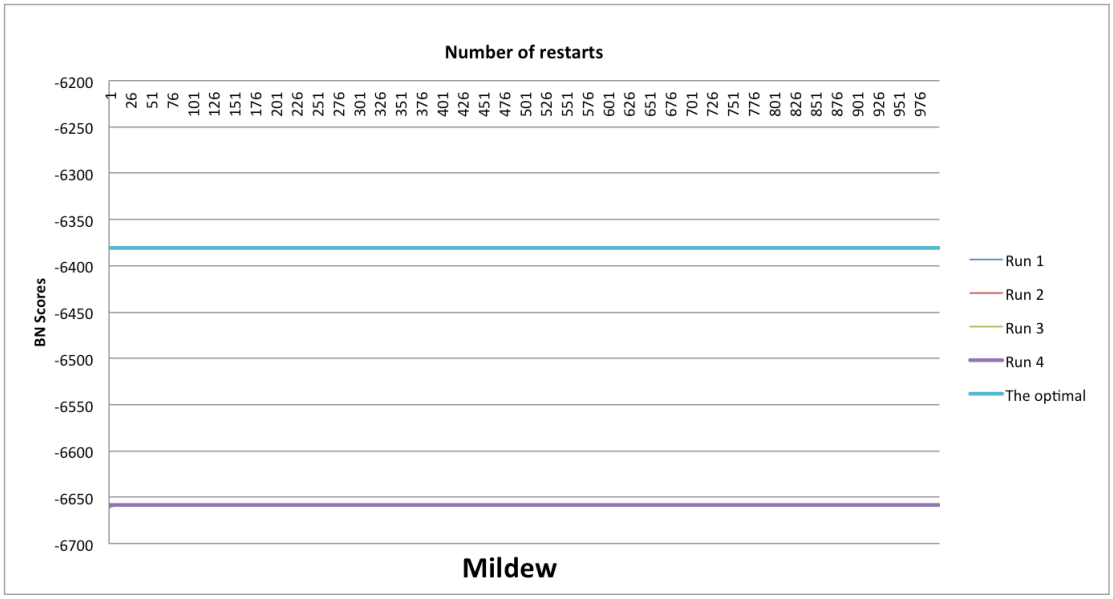

Figure 31: Mildew network without prior knowledge

Figure 29 shows the results of applying HC without prior knowledge for synthetic data generated by the Mildew network when we plot the score of the best network found so far against the number of restarts and each run has 1,000 random restarts. As shown, HC without incorporating prior knowledge always found the same network. This is mainly explained by the fact that local scores of different parent sets tend to be very similar.

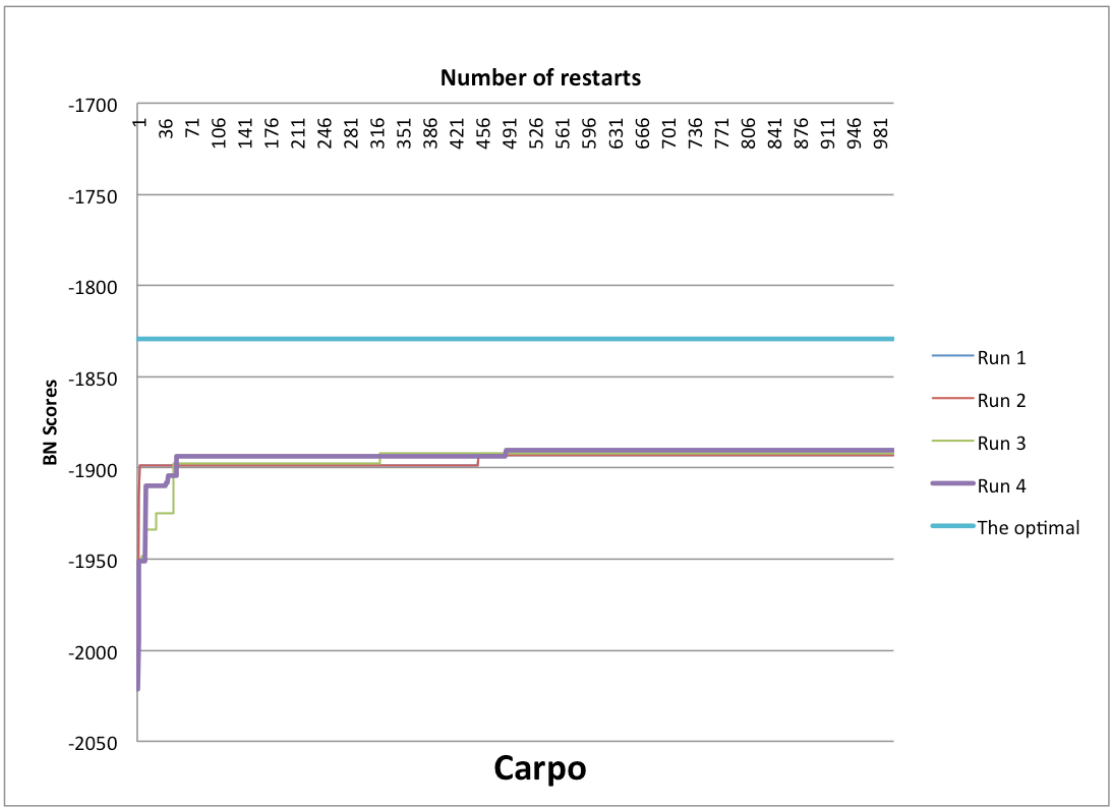

Figure 32: Carpo network without prior knowledge.

Figure 30 shows the results of applying HC without prior knowledge for synthetic data generated by the Carpo network. We plotted the results of the best network found so far against the number of restarts. As shown, HC without incorporating prior knowledge: the best network was found at 53, and then a better network was found at 488.

6.2.1 Arrows Which Have To Be There $A \leftarrow B$

With this prior knowledge, an arrow can be added to a particular child based on the given parents. A child is selected first, and then a decision is made in terms of which parents to add. The HCPK algorithm allows the network for many possible parents to be learnt. For example, take $A \leftarrow B$, where B must be a parent of A. Assume that A is the child specified by the user and that B is the parent specified by the user knowledge (discussed in section 5.3.2).

With entire parent sets specified, * indicates the child variable where there are v-structures (Please see Appendix-B for further details). BDe scores of learned Bayesian network are mapped against items of prior knowledge. BDe scores of the network of our algorithm are found with prior knowledge and without. Adding more prior knowledge will have a greater effect on the learning algorithm, which can sometimes achieve an optimal network. Also, in terms of the effect of prior knowledge for bigger dataset sizes, if users specify the entire parent set, as shown, prior knowledge has typically an effect on the learning.

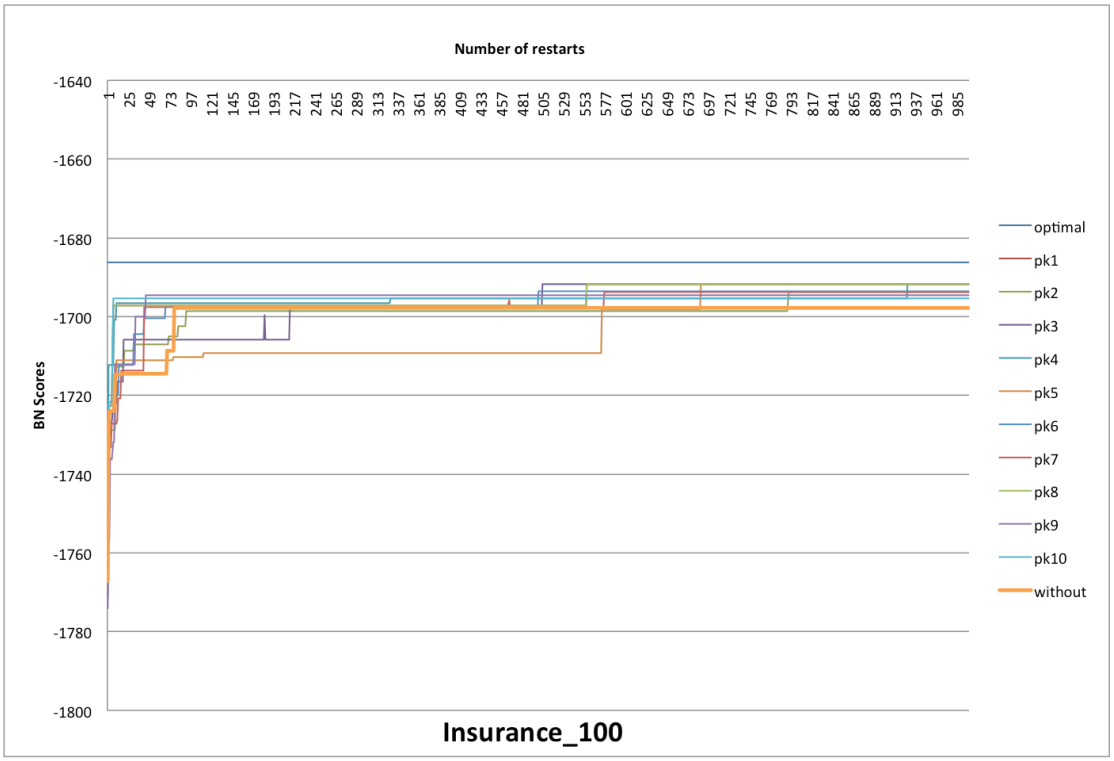

Figure 33: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Insurance100.

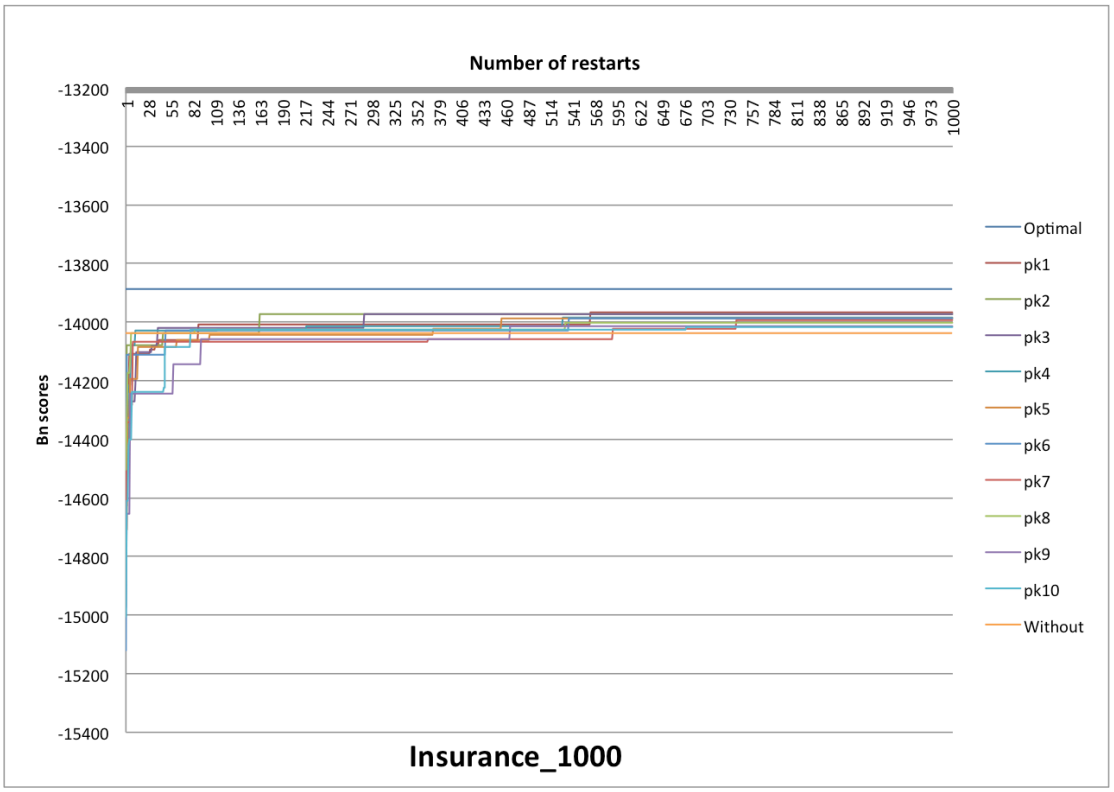

Figure 34: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Insurance 1000.

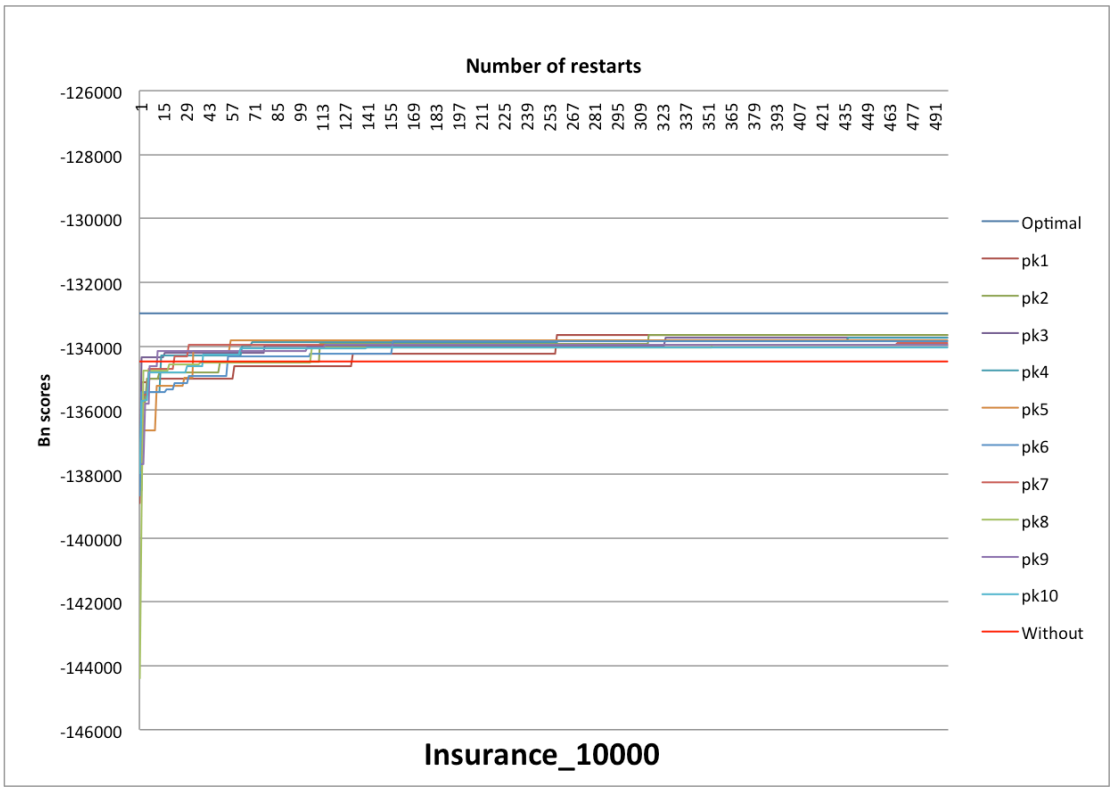

Figure 35: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Insurance 10000.

Figure 36: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Mildew 100.

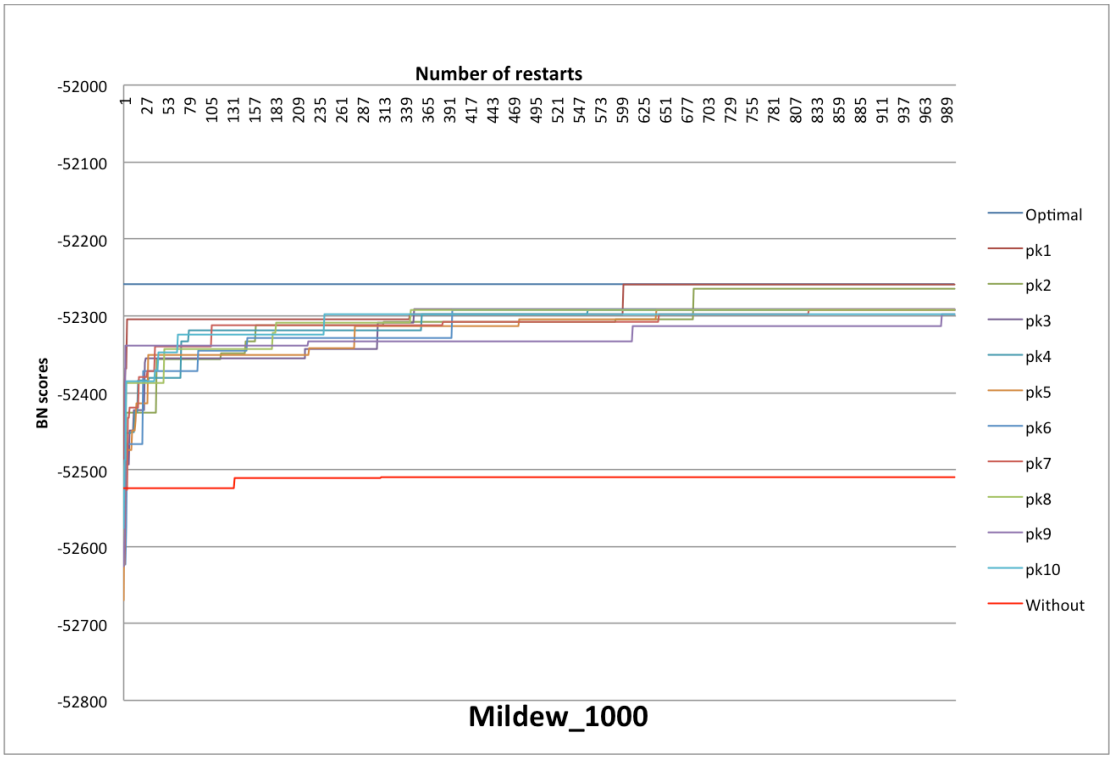

Figure 37: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Mildew 1000.

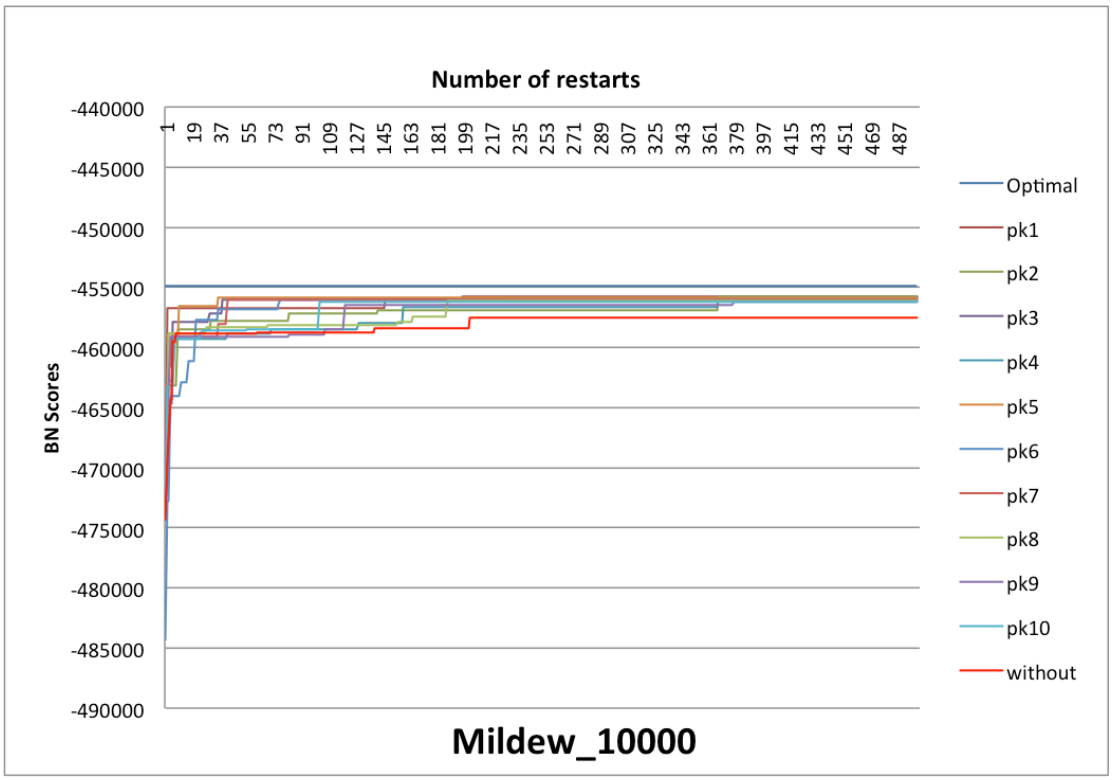

Figure 38: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Mildew 10000.

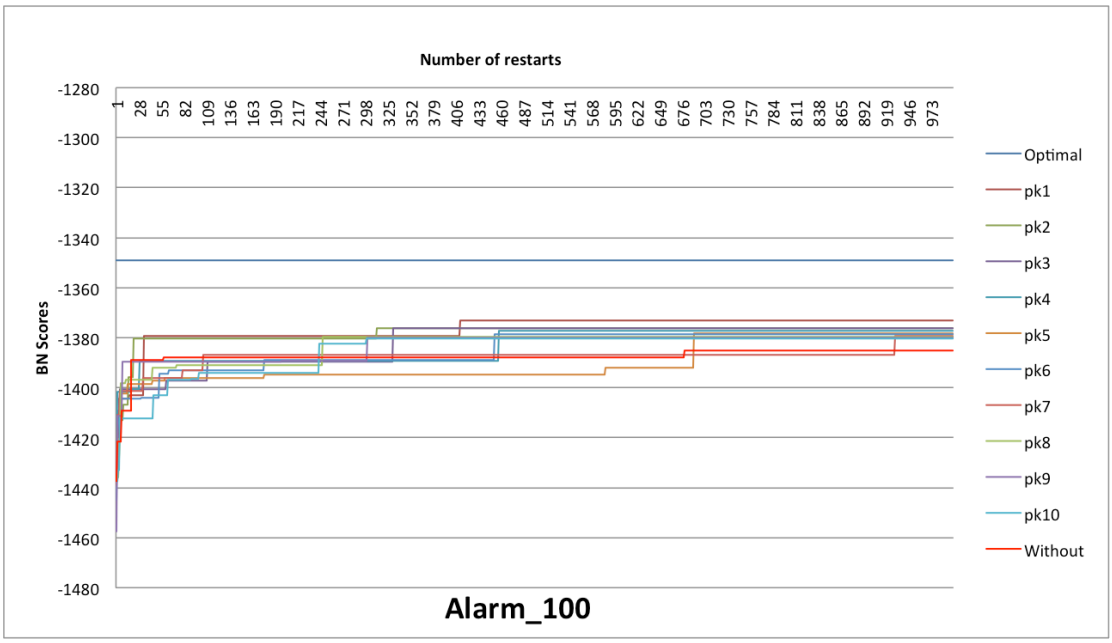

Figure 39: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Alarm 100.

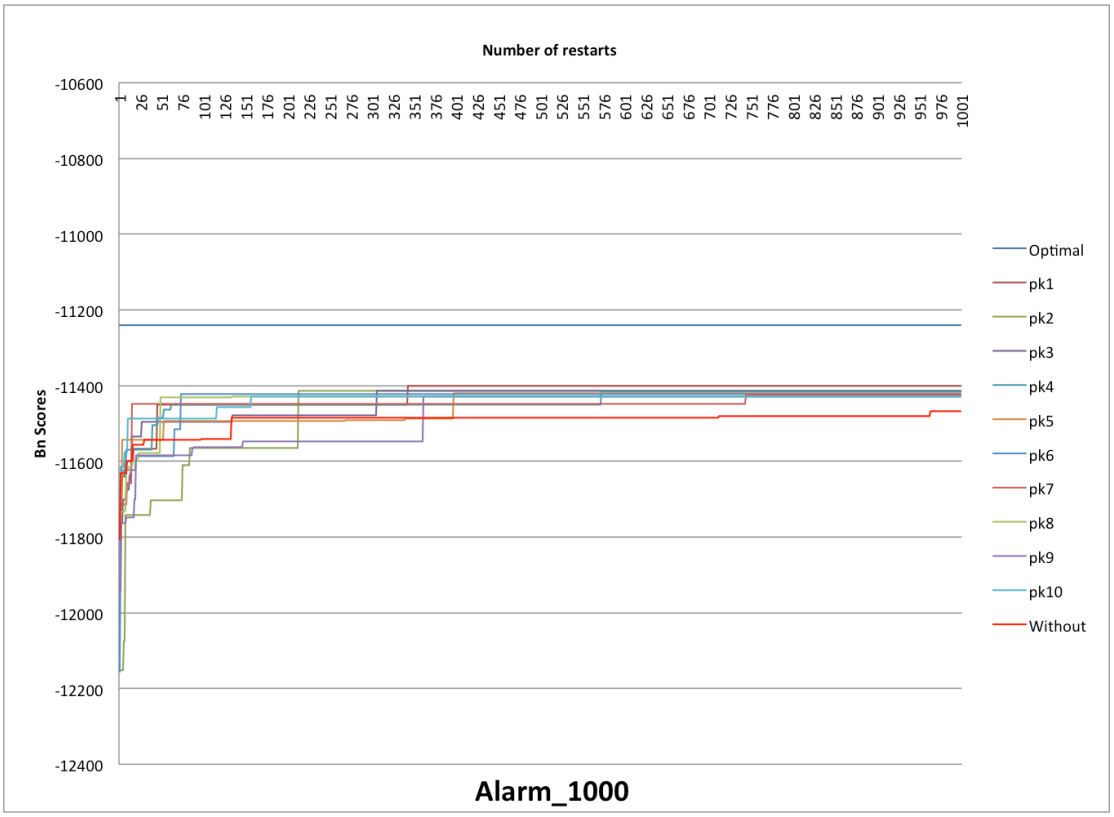

Figure 40: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Alarm 1000.

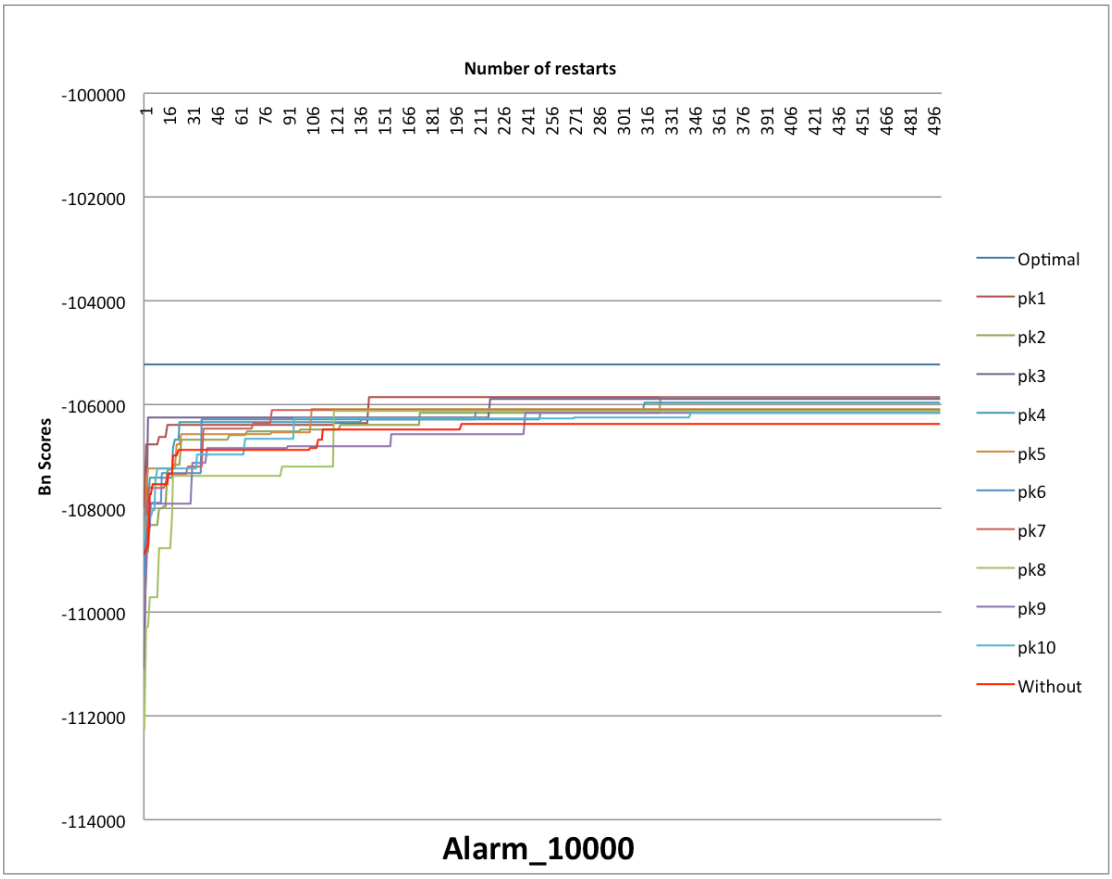

Figure 41: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Alarm 10000.

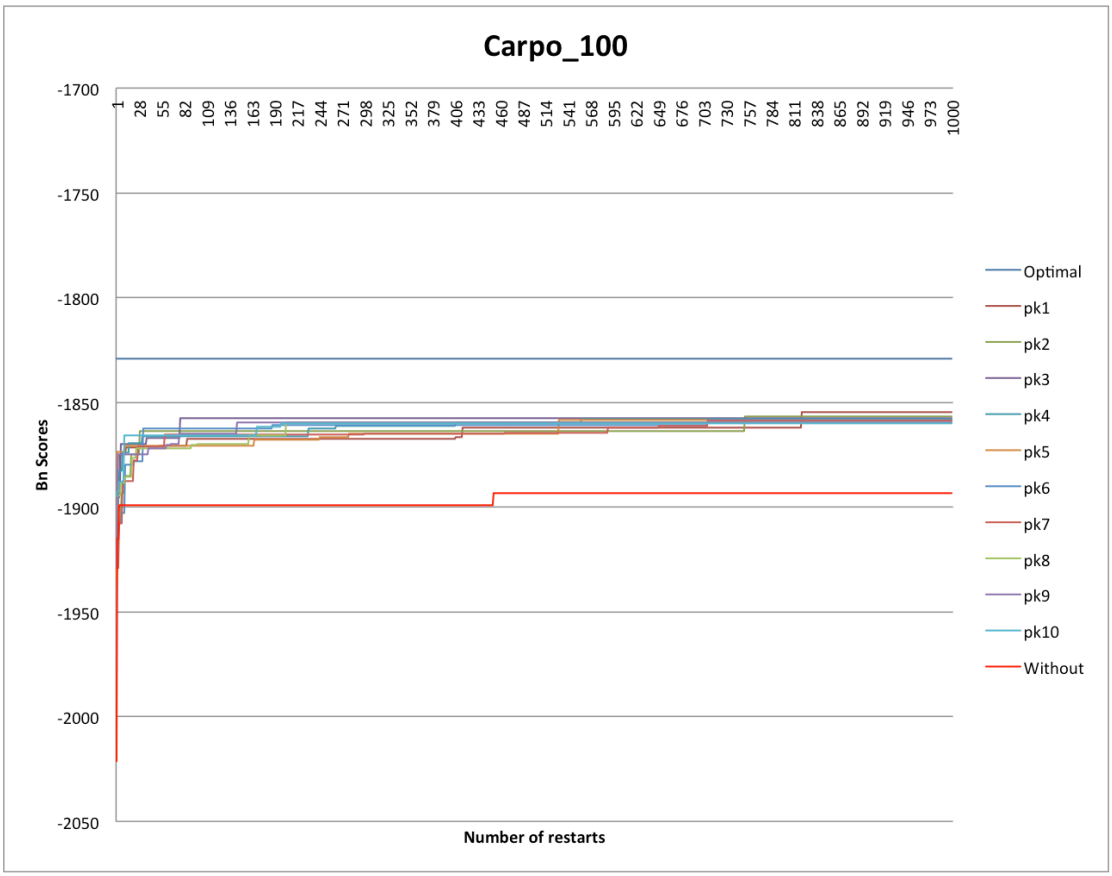

Figure 42: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Carpo 100.

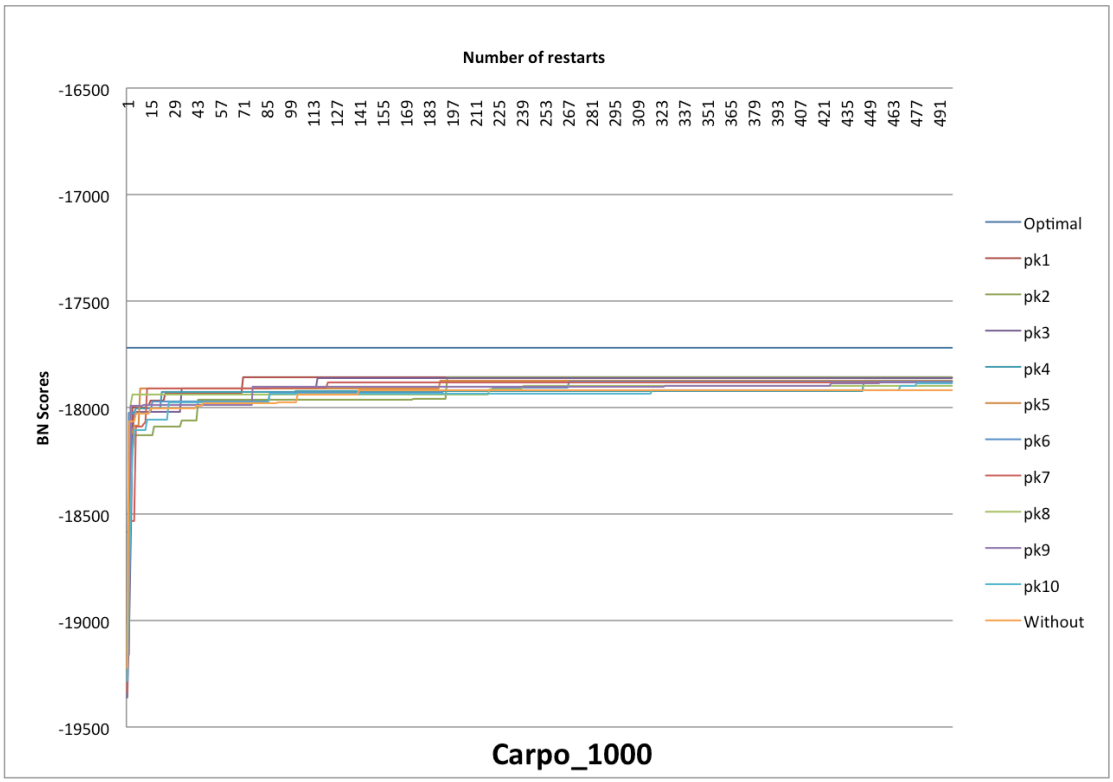

Figure 43: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Carpo 1000.

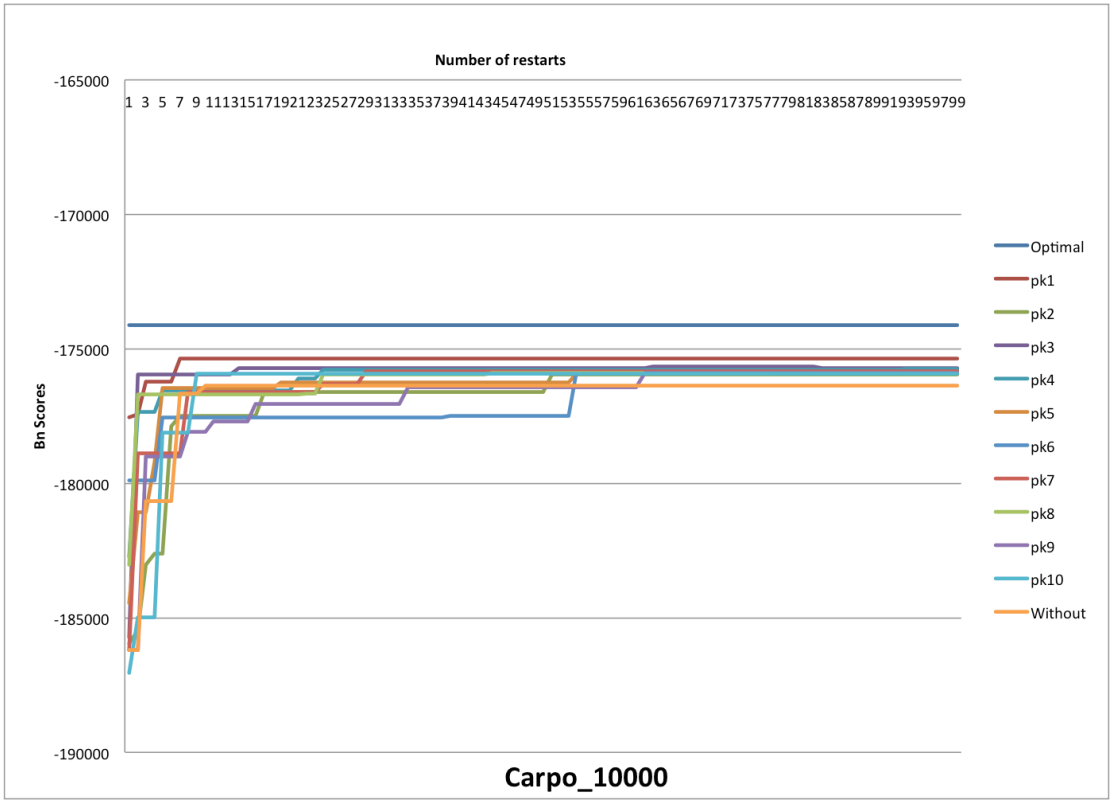

Figure 44: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Carpo 10000.

For prior knowledge, users can specify entire parent sets, or part of them, with the appropriate arrows. If users specify entire parent sets, this has a greater effect on learning. It is also interesting to note that specifying a parent set of a variable where there was a v-structure had a beneficial effect on learning. If we add more prior knowledge, this has a greater effect on the learning algorithm, which sometimes achieves the optimal network.

Even for larger dataset sizes, if users specify the entire parent set, then prior knowledge has an effect on the learning. However, the effect of prior knowledge if users specify part of a parent set is similar to the result seen when users specify the entire parent set.

6.2.2 Ancestor Relation

This section shows the results of incorporating ancestor relation prior knowledge. For example, take the prior knowledge $B \dashleftarrow A$ which indicates that A must be an ancestor of B. Assume that B is the specified child by the user and that A is the specified ancestor by the user (discussed in section 5.3.3).

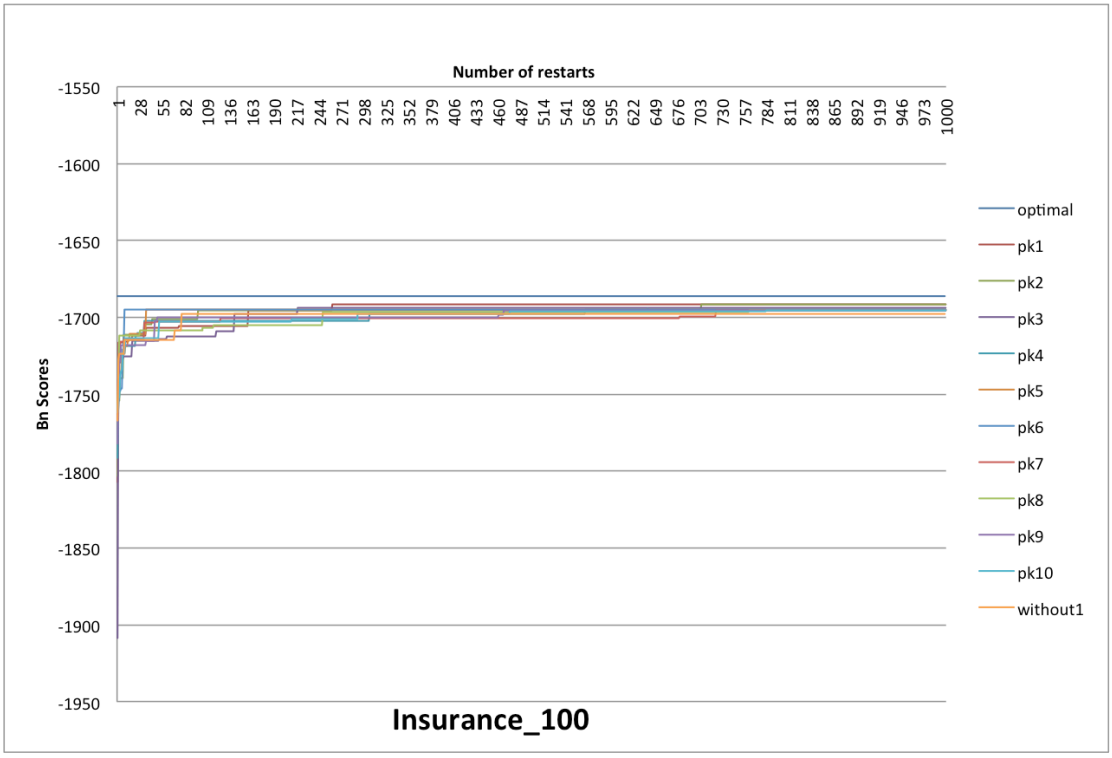

Figure 45: The results of applying HCPK, ancestors relation, for synthetic data generated by the Insurance 100.

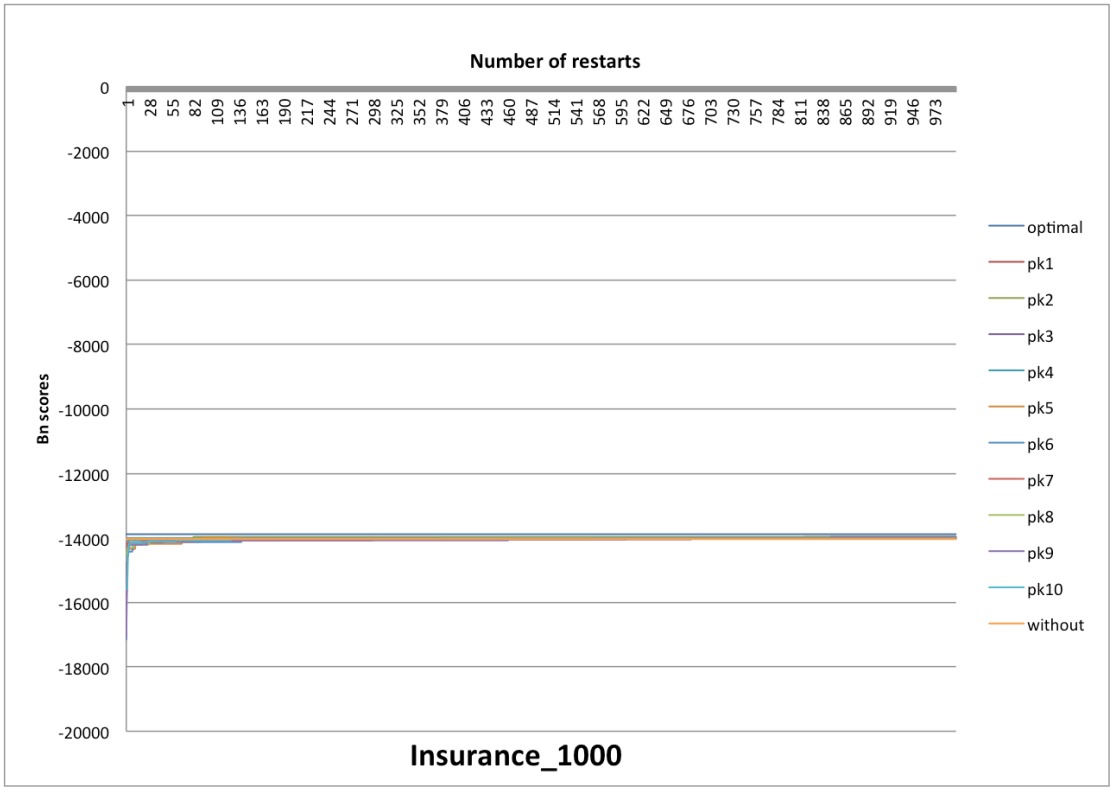

Figure 46: The results of applying HCPK, ancestors relation, for synthetic data generated by the Insurance 1000.

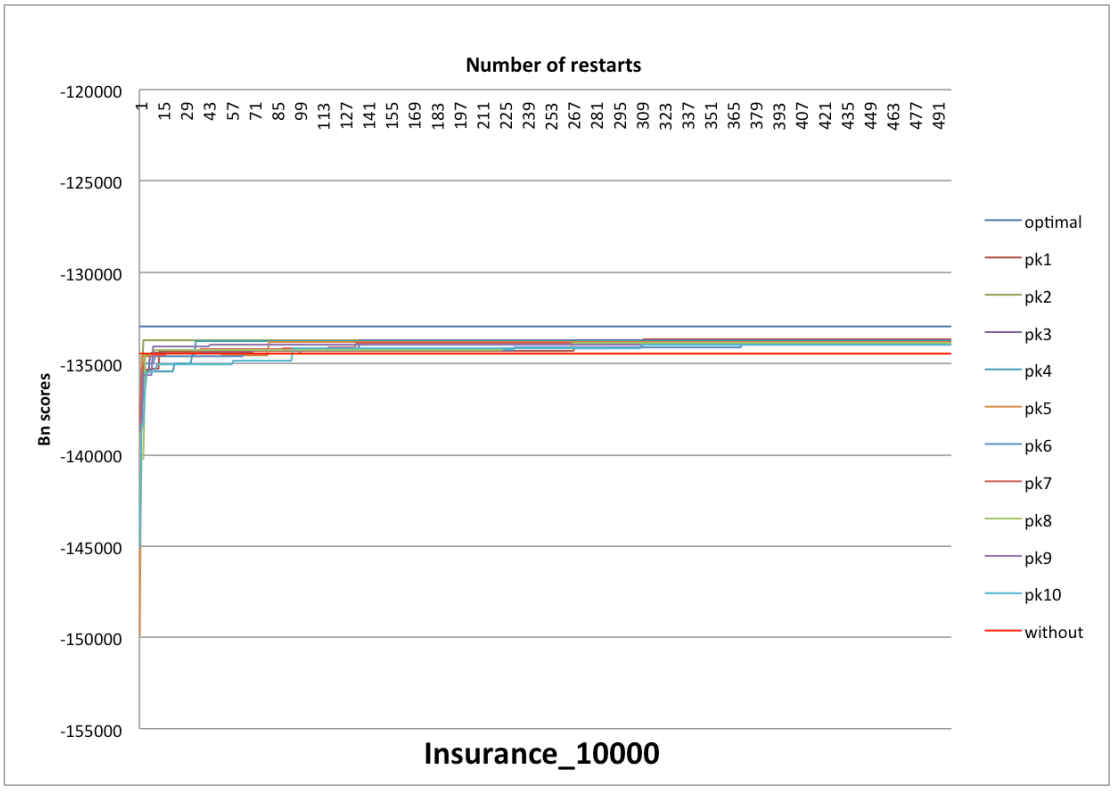

Figure 47: The results of applying HCPK, ancestors relation, for synthetic data generated by the Insurance 10000.

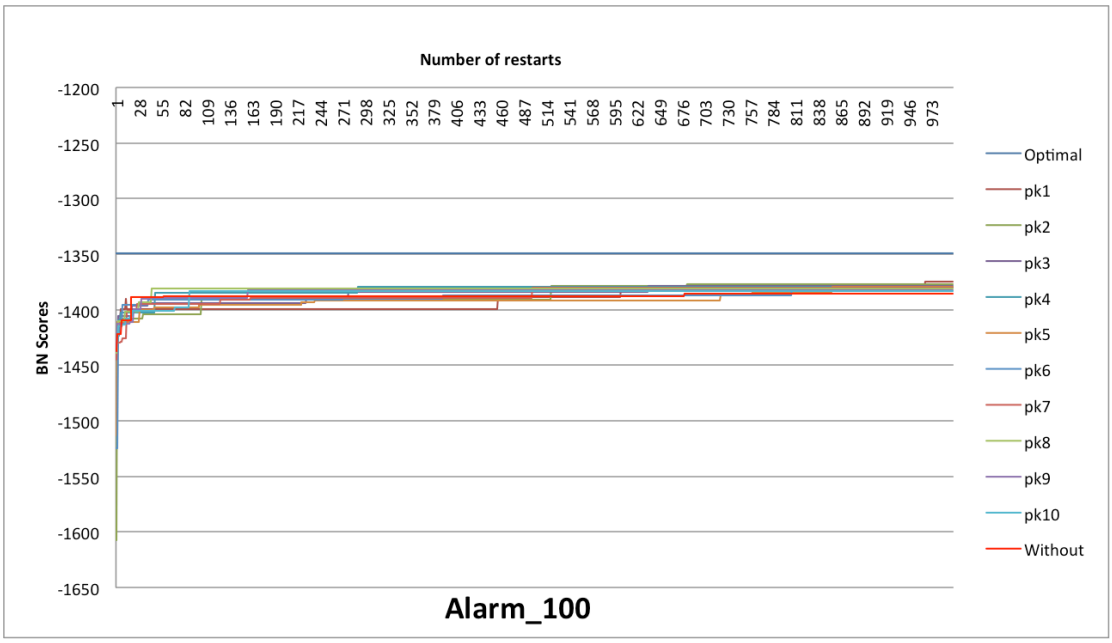

Figure 48: The results of applying HCPK, ancestors relation, for synthetic data generated by the Alarm 100.

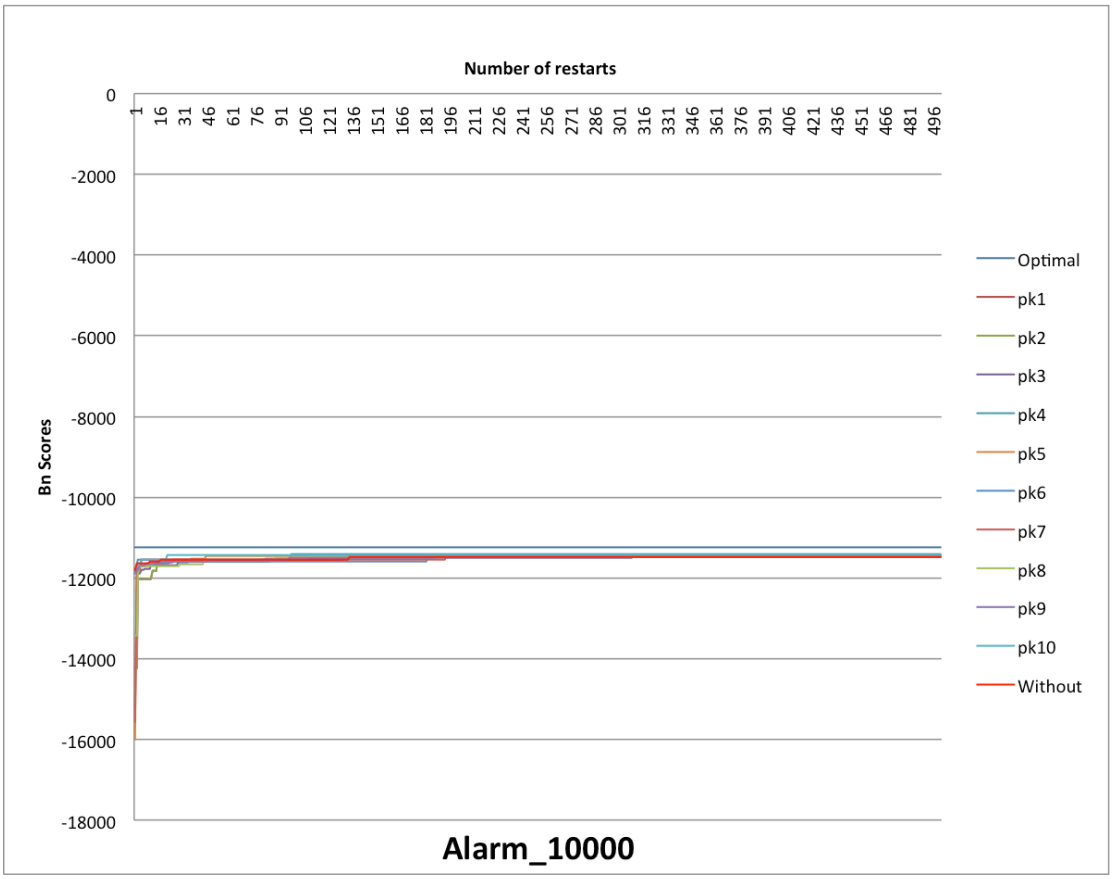

Figure 49: The results of applying HCPK, ancestors relation, for synthetic data generated by the Alarm 1000.

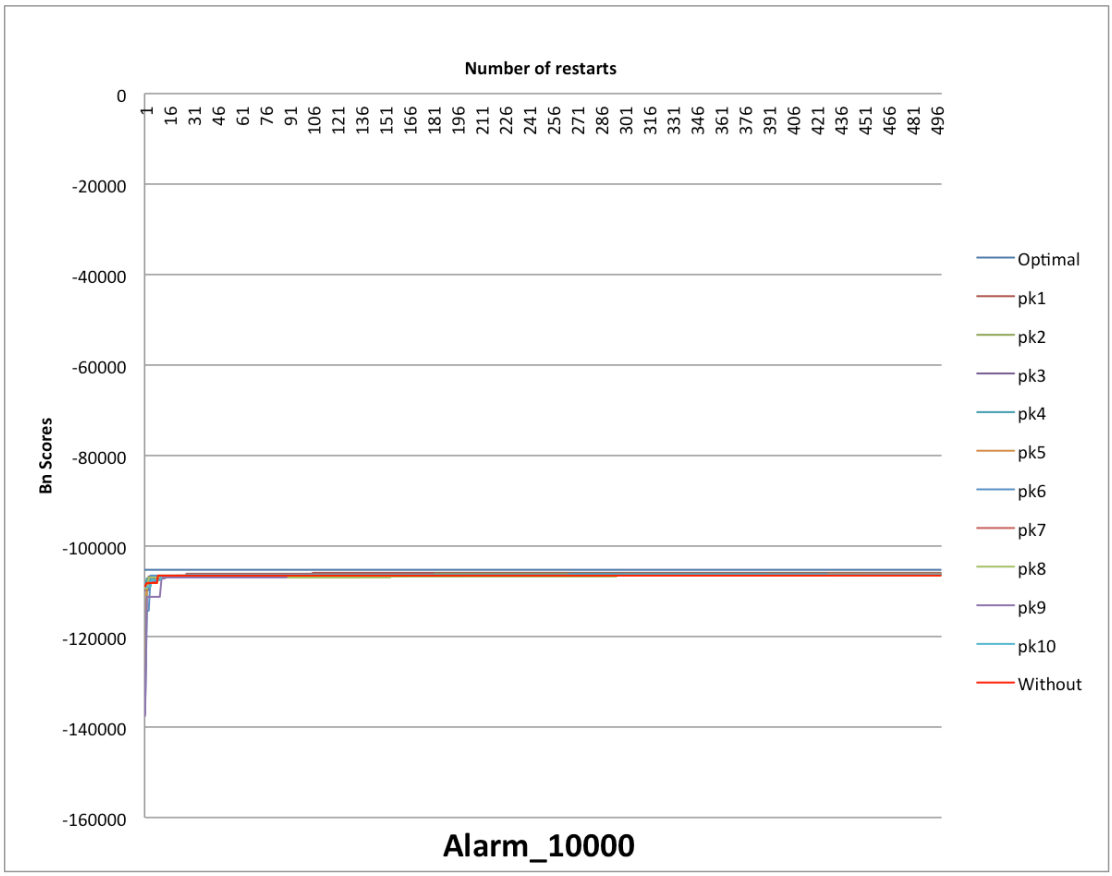

Figure 50: The results of applying HCPK, ancestors relation, for synthetic data generated by the Alarm 10000.

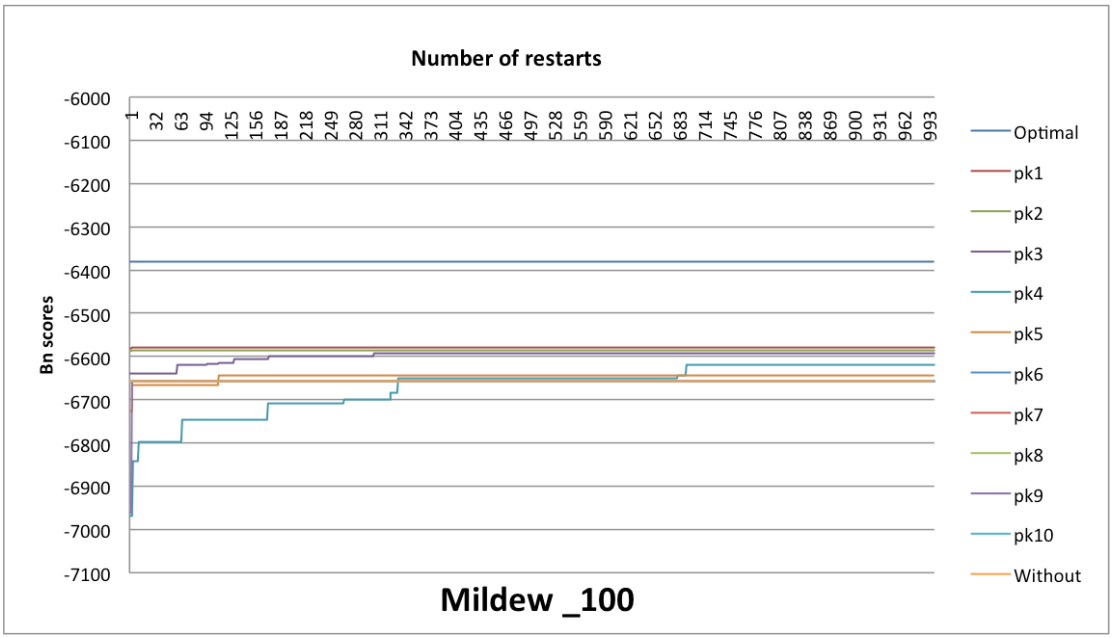

Figure 51: The results of applying HCPK, ancestors relation, for synthetic data generated by the Mildew 100.

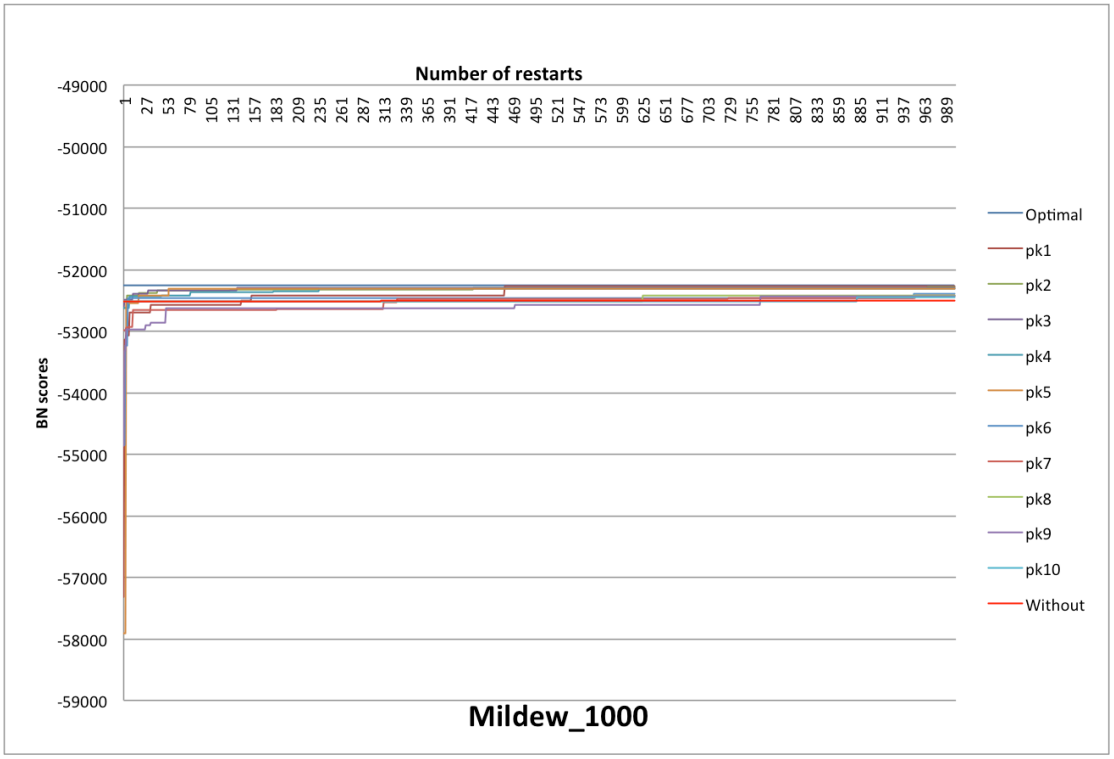

Figure 52: The results of applying HCPK, ancestors relation, for synthetic data generated by the Mildew 1000.

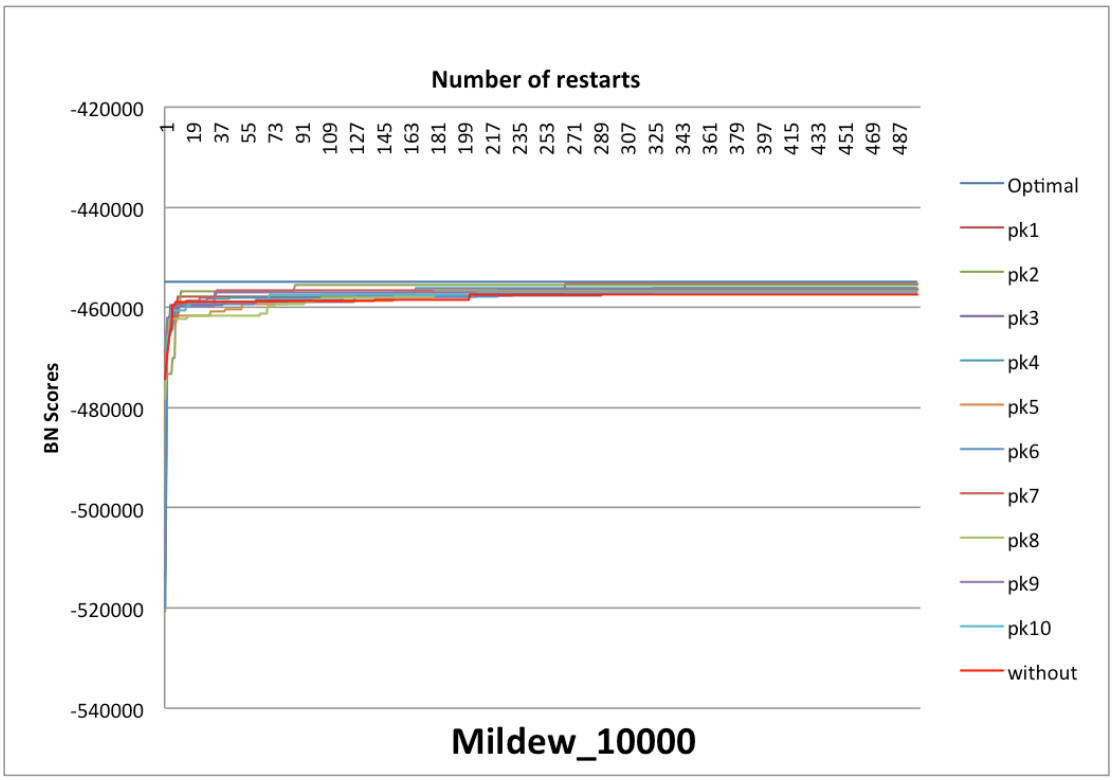

Figure 53: The results of applying HCPK, ancestors relation, for synthetic data generated by the Mildew 10000.

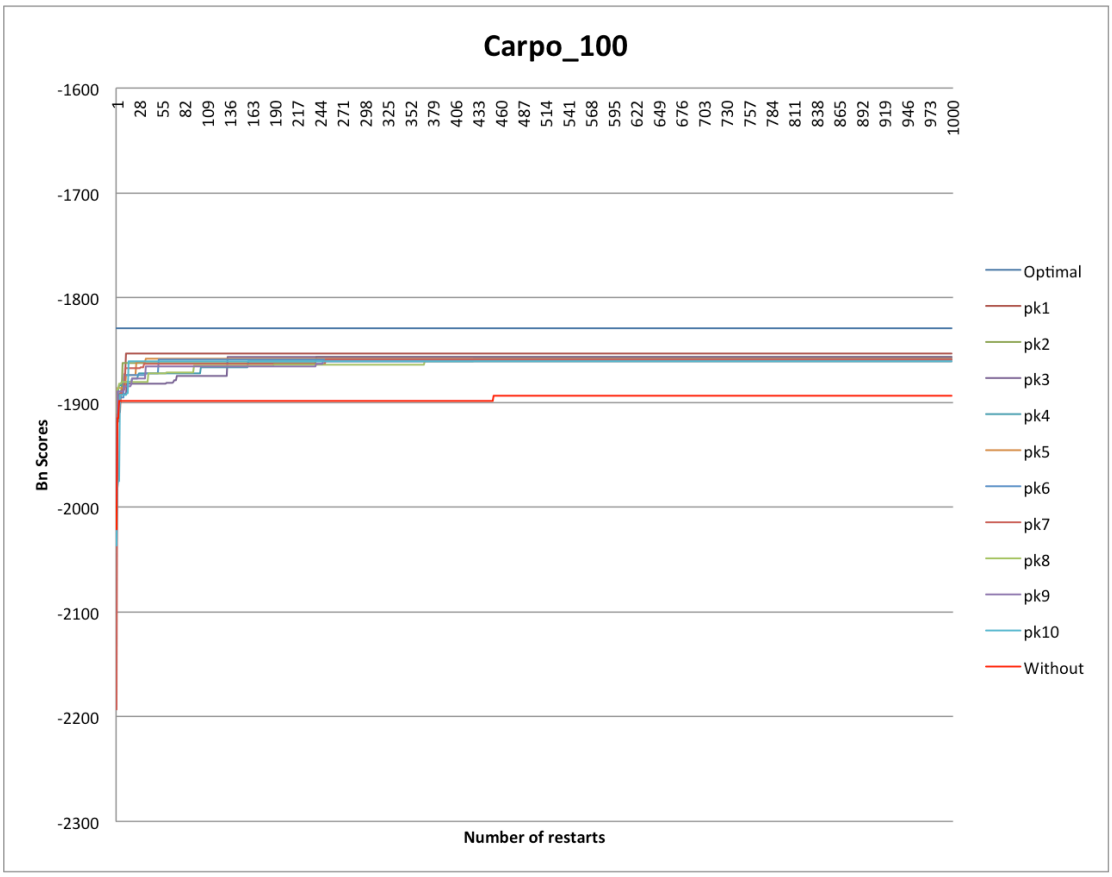

Figure 54: The results of applying HCPK, ancestors relation, for synthetic data generated by the Carpo 100.

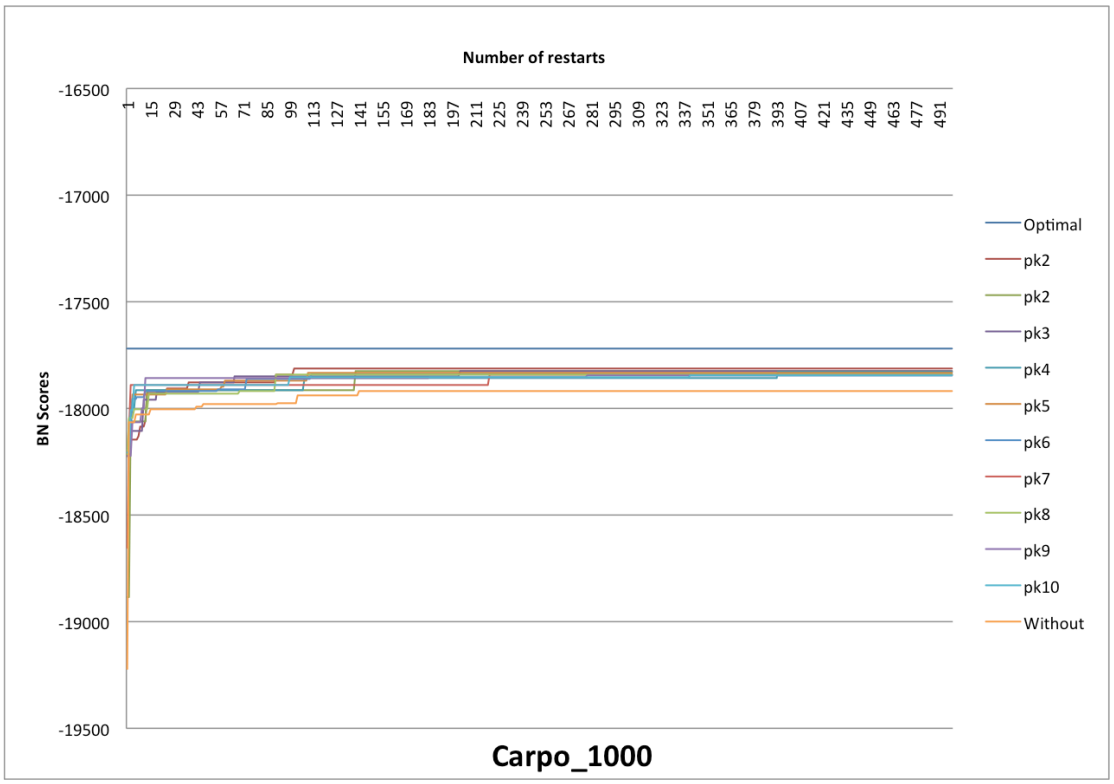

Figure 55: The results of applying HCPK, ancestors relation, for synthetic data generated by the Carpo 1000.

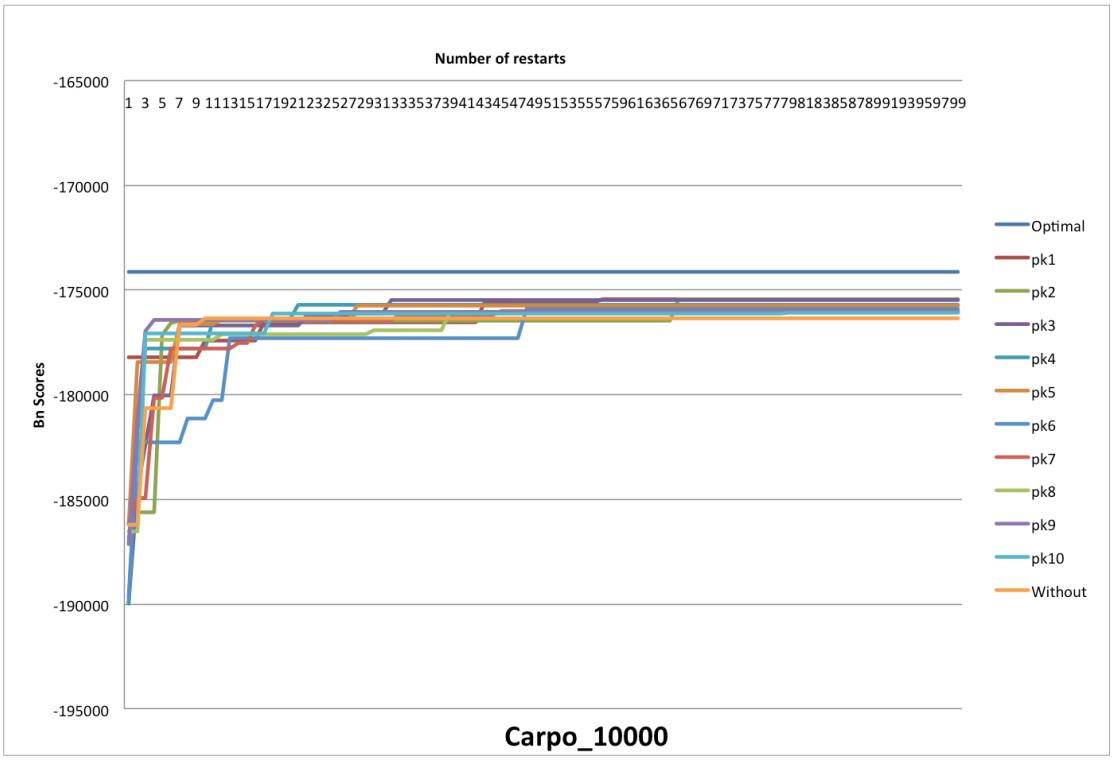

Figure 56: The results of applying HCPK, ancestors relation, for synthetic data generated by the Carpo 10000.

With respect to ancestor relation, these tables show that prior knowledge typically has an effect on learning a Bayesian network (Please see details in Appendix-C). However, the algorithm retains records of the selected parent sets for any single restart in order to avoid repetition later in the search. Therefore, there is a trade-off between the highest scoring network and a network consistent with the user's prior knowledge. As a result, it should be noted that learning is less affected by prior knowledge for some datasets.

In Figure 49, we noticed that ancestor information that is usually beneficial turned out not to be in this case. This is primarily explained by the fact that local scores of different parent sets tend to be very similar. However, it is more difficult for prior knowledge to help us obtain a high-scoring network in any file where the local scores tend to be quite similar to each other, such as, for example, mildew datasets.

6.2.3 Conditional Independence

Dependencies and independencies are the main issues in a probability distribution as discussed in section 5.3.4. Local independencies in Bayesian networks are where each node is independent of its non-descendants, given its parents. Global independencies are derived from d-separation, which helps to ensure that specific sets of independencies ($A \perp B \mid Z$) hold in a distribution, so that a variable A is conditionally independent of a particular variable B, given its variable Z. In this section, we show the result of incorporating the conditional independence prior knowledge into the developed learning algorithm using two different approaches.

6.2.3.1 Conditional independence checks approach

In this section we show the effect of including conditional independence prior knowledge, we used a d-Separation algorithm for each move in the HCPK algorithm. Therefore, for each additional move, the algorithm creates a *temporary graph* that contains the current graph and the possible parent set. It then checks this *temporary graph* for conditional independence. If the possible parent sets do not satisfy the conditional independence checks, it is not considered. Using this approach, we continue to build the graph by conducting these early checks and, eventually, end up with a network that meets the user's prior knowledge (discussed in section 5.3.4.2, algorithm 4).

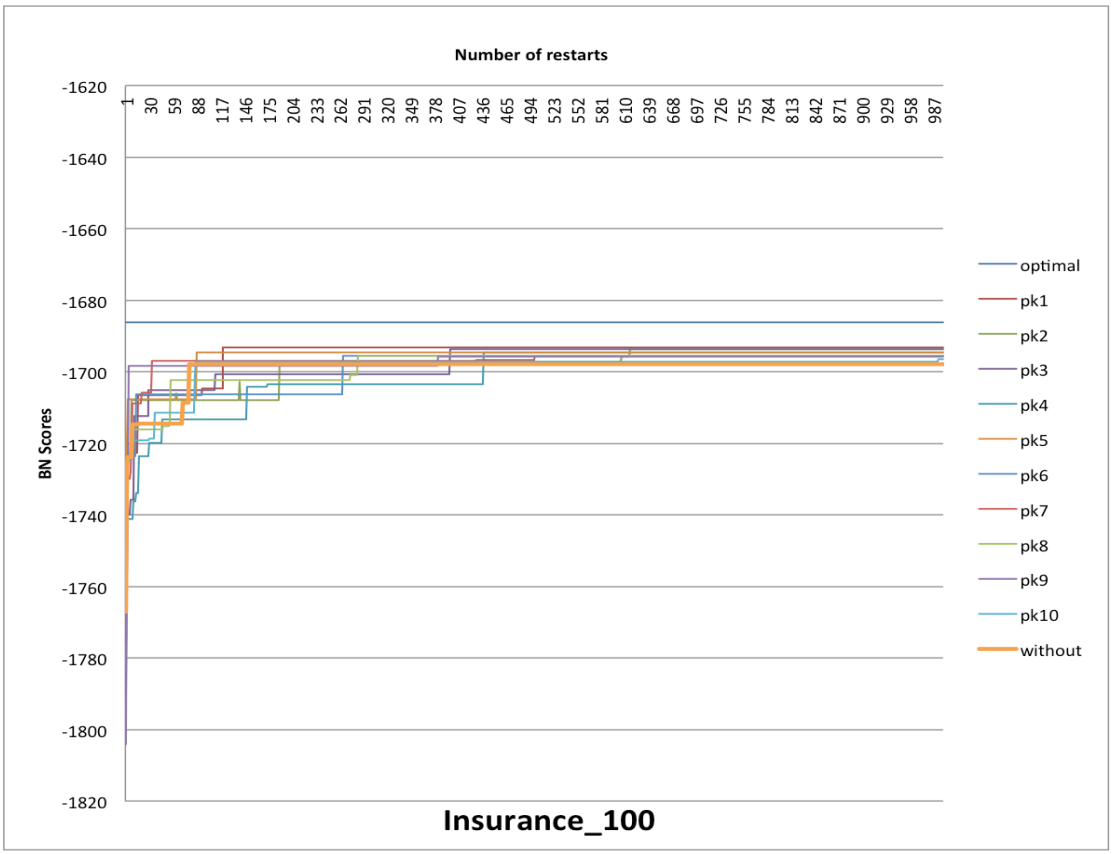

Figure 57: The results of applying HCPK, conditional independence, for synthetic data generated by the Insurance 100.

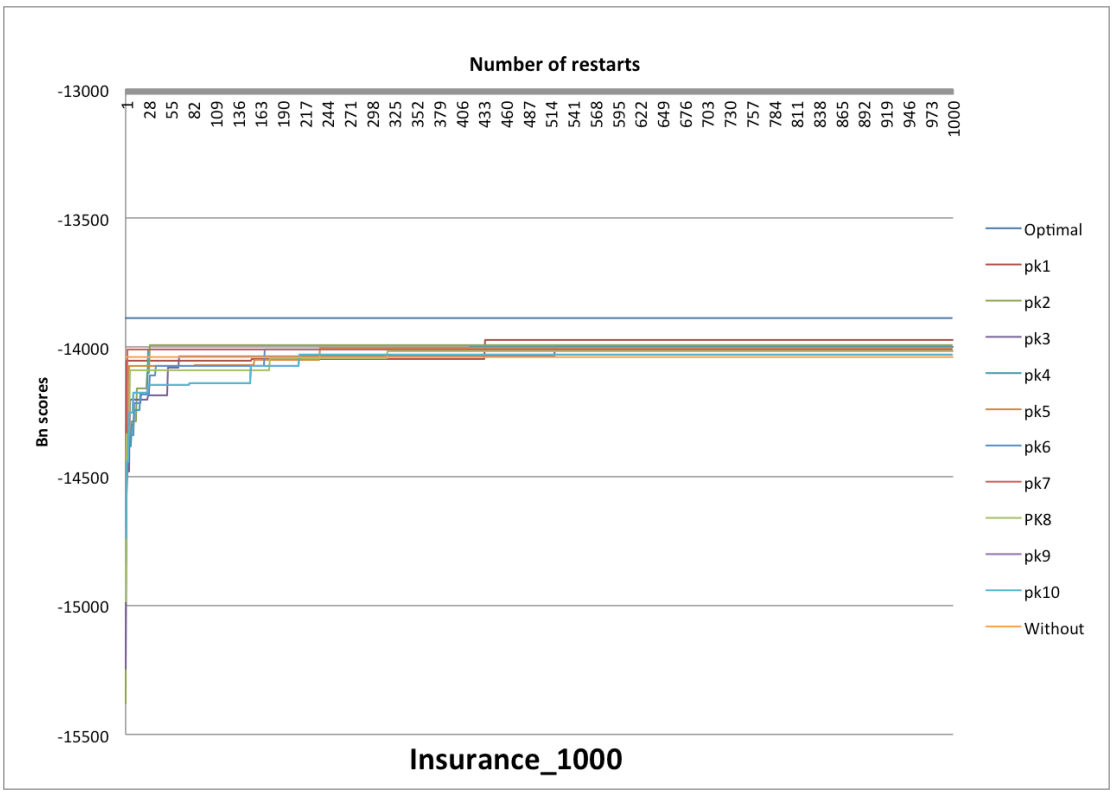

Figure 58: The results of applying HCPK, conditional independence, for synthetic data generated by the Insurance 1000.

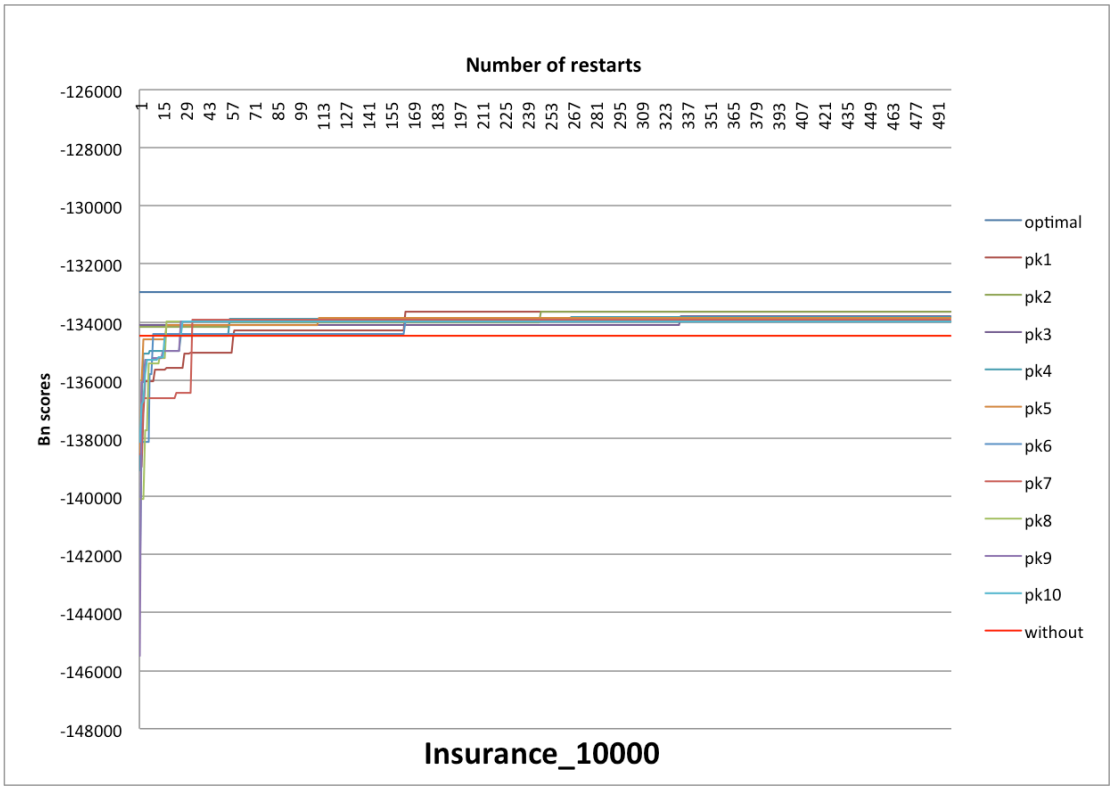

Figure 59: The results of applying HCPK, conditional independence, for synthetic data generated by the Insurance 10000.

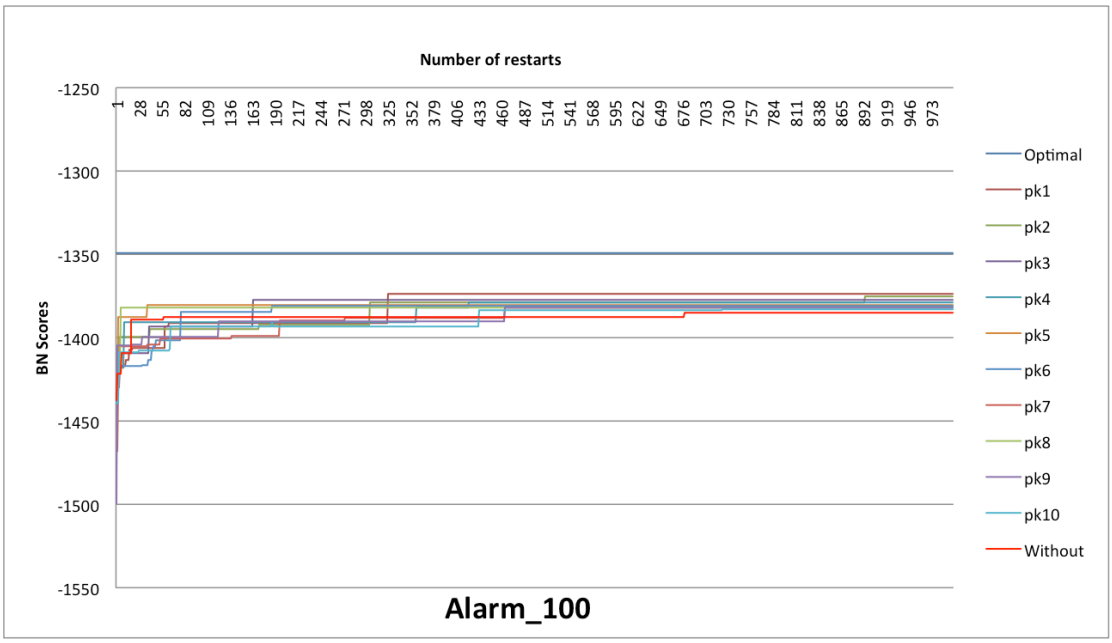

Figure 60: The results of applying HCPK, conditional independence, for synthetic data generated by the Alarm 100.

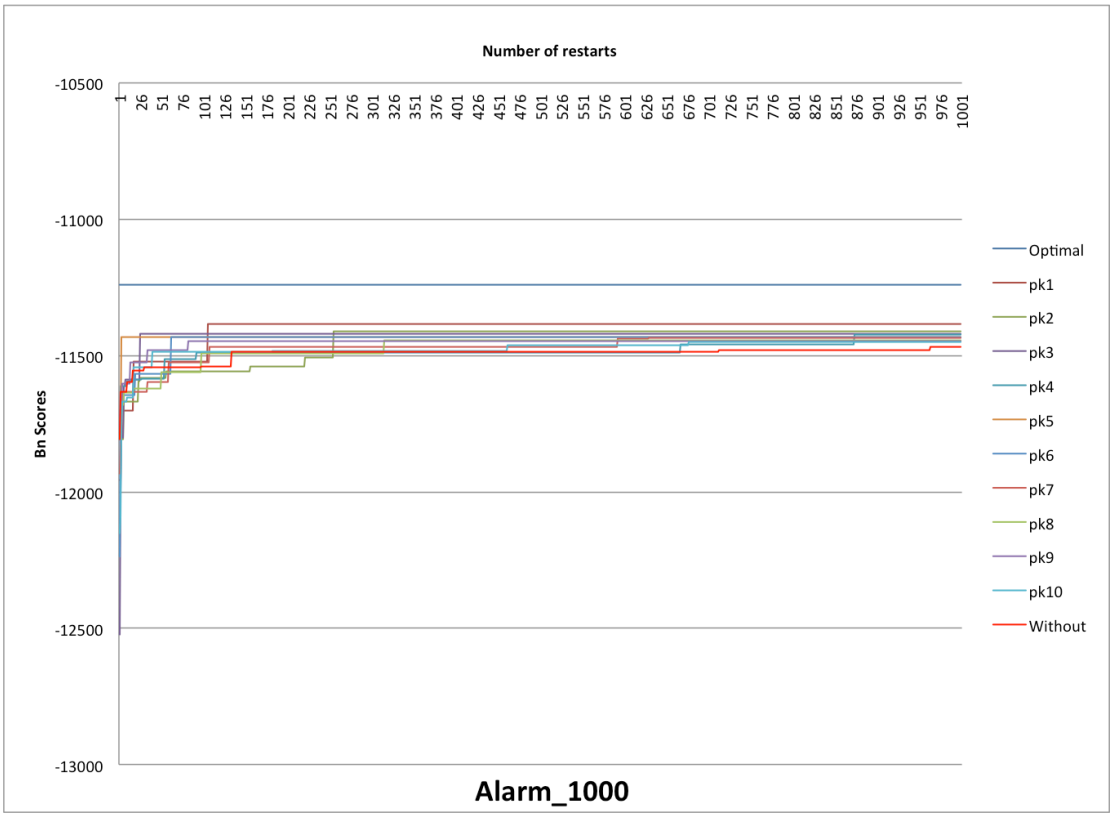

Figure 61: The results of applying HCPK, conditional independence, for synthetic data generated by the Alarm 1000.

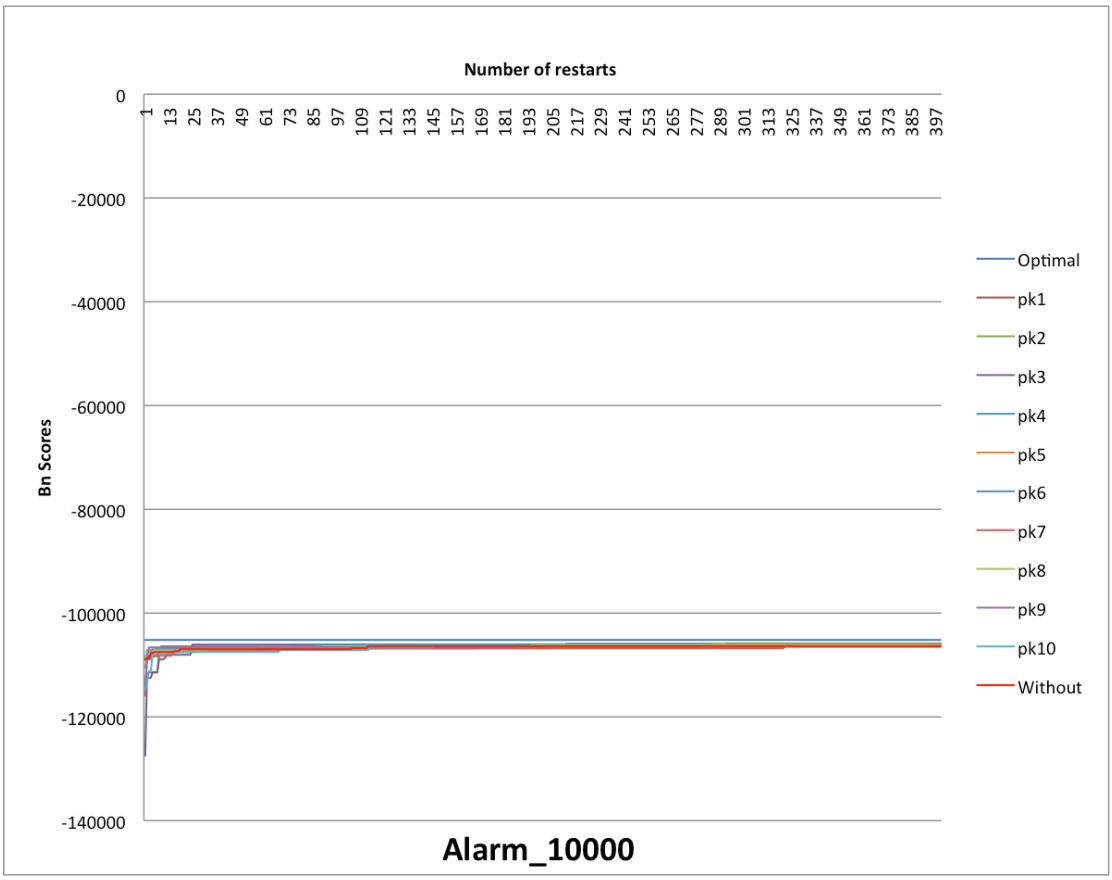

Figure 62: The results of applying HCPK, conditional independence, for synthetic data generated by the Alarm 10000.

Figure 63: The results of applying HCPK, conditional independence, for synthetic data generated by the Mildew 100.

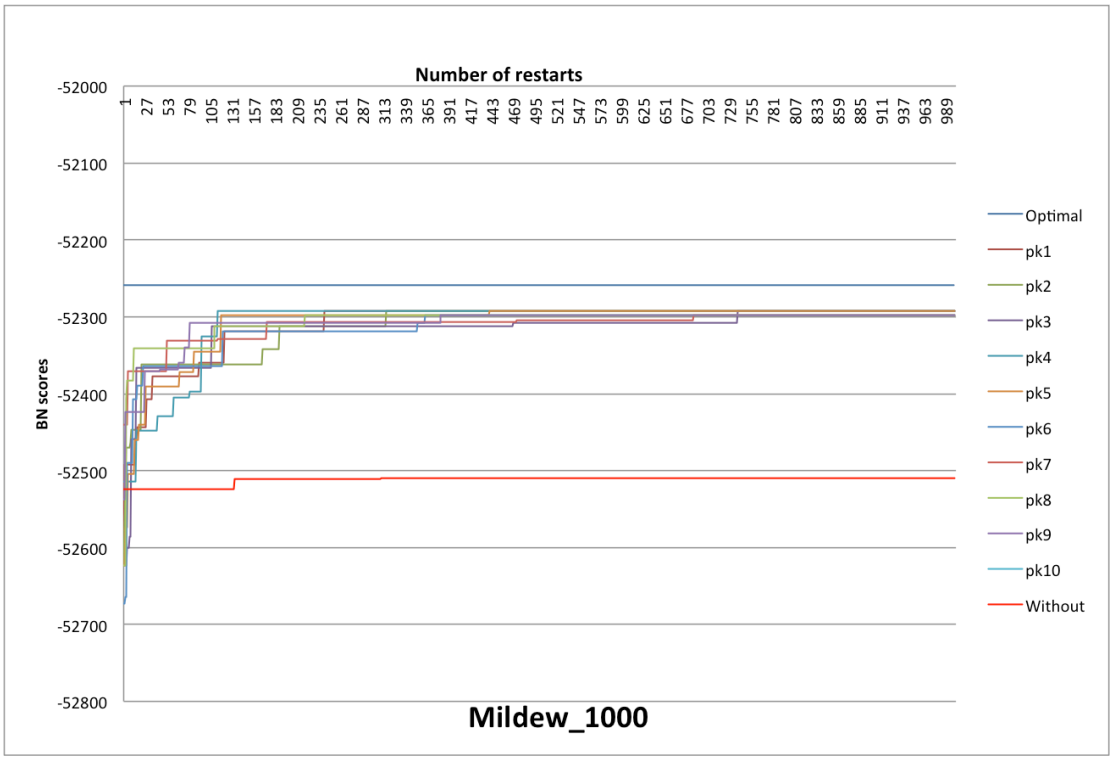

Figure 64: The results of applying HCPK, conditional independence, for synthetic data generated by the Mildew 1000.

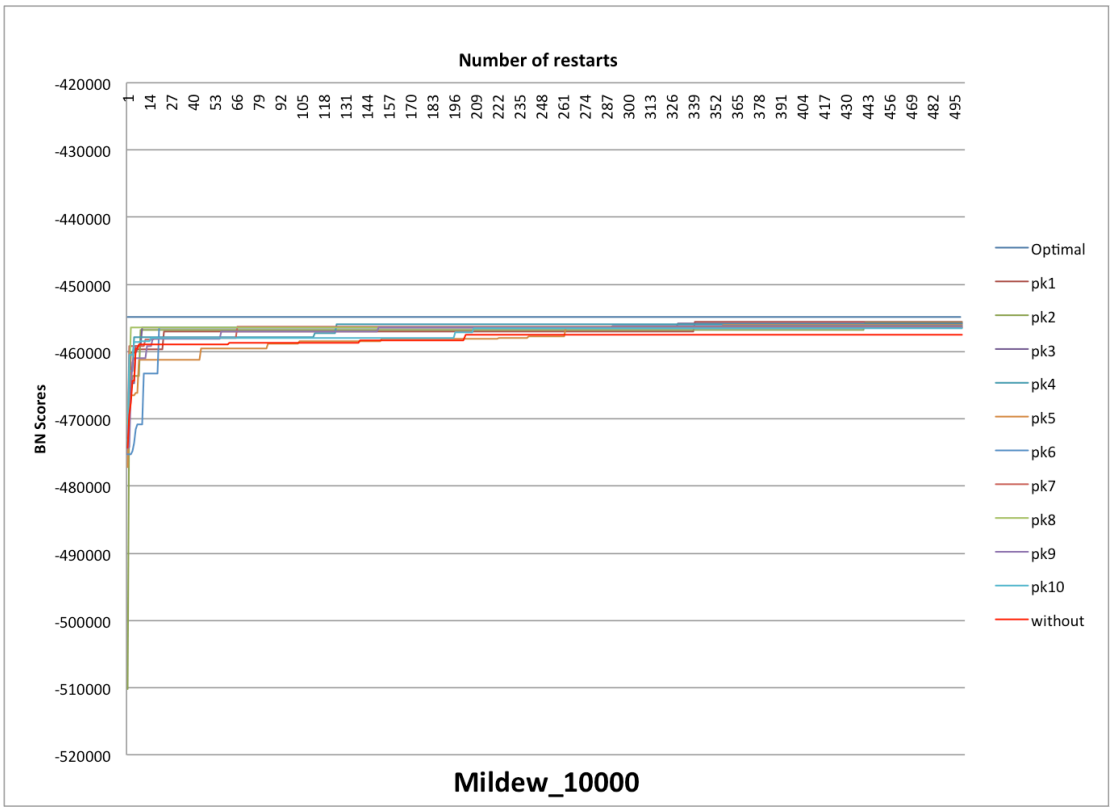

Figure 65: The results of applying HCPK, conditional independence, for synthetic data generated by the Mildew 10000.

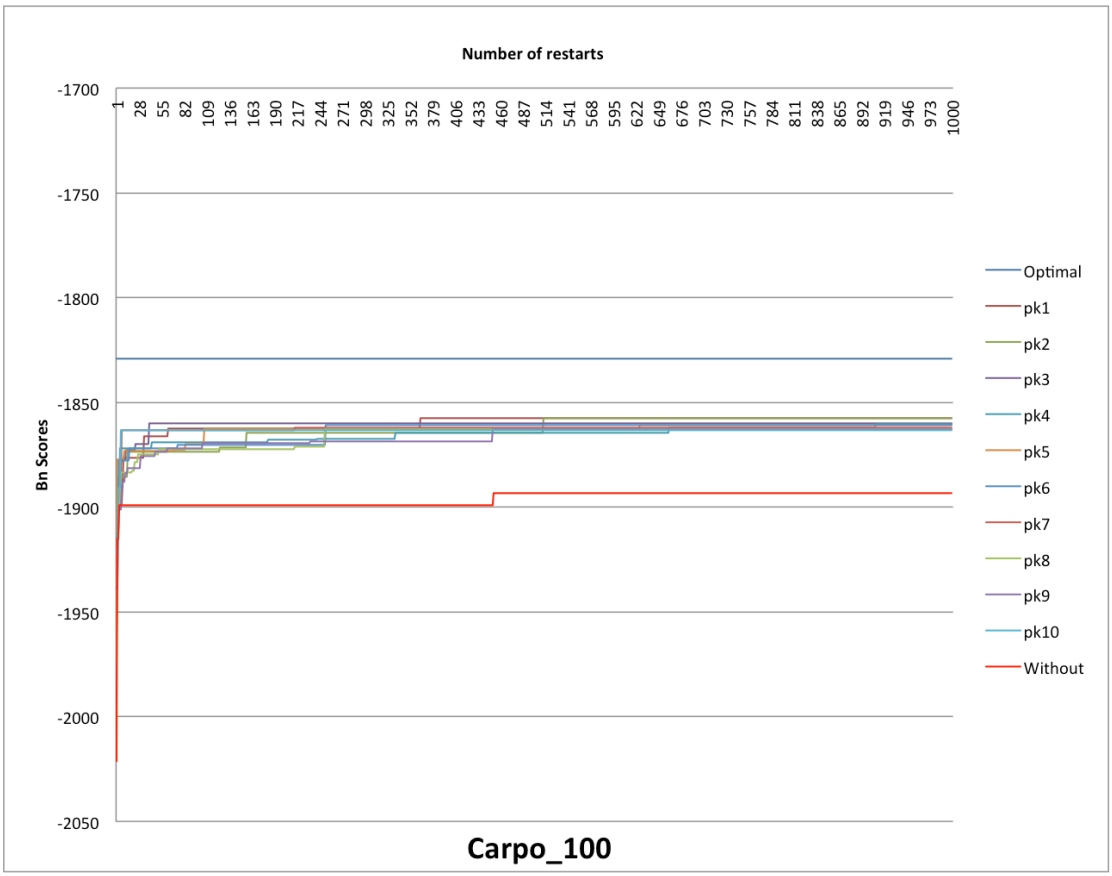

Figure 66: The results of applying HCPK, conditional independence, for synthetic data generated by the Carpo 100.

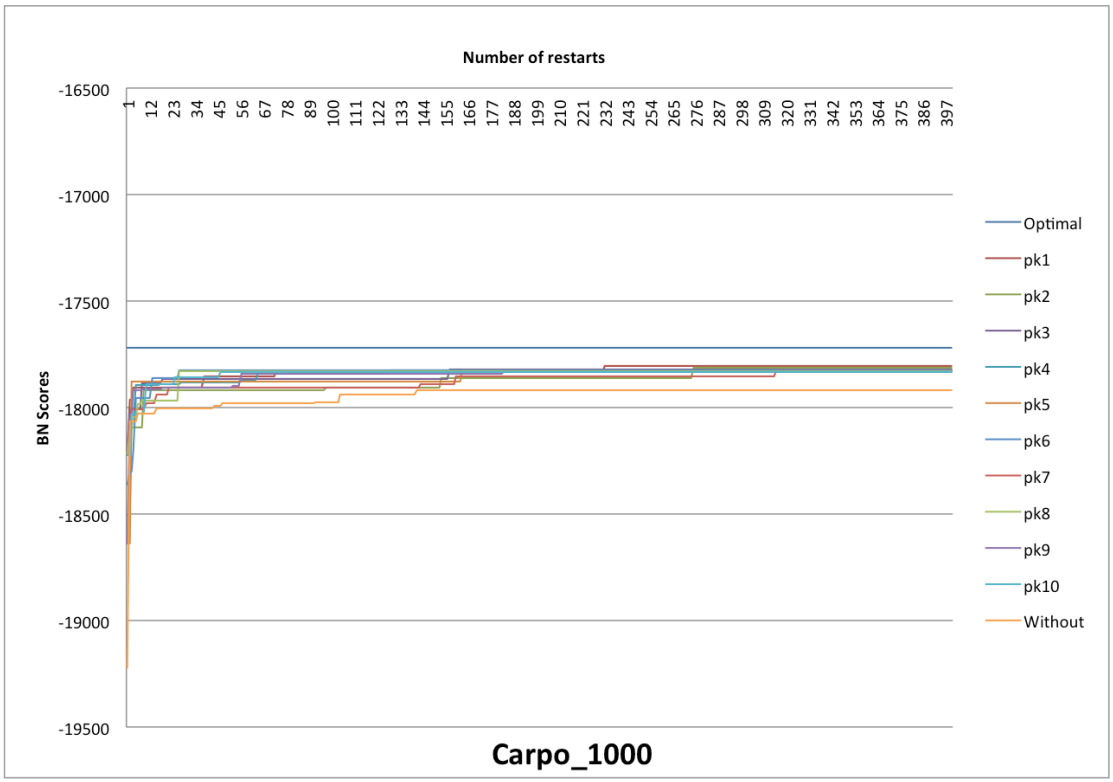

Figure 67: The results of applying HCPK, conditional independence, for synthetic data generated by the Carpo 1000.

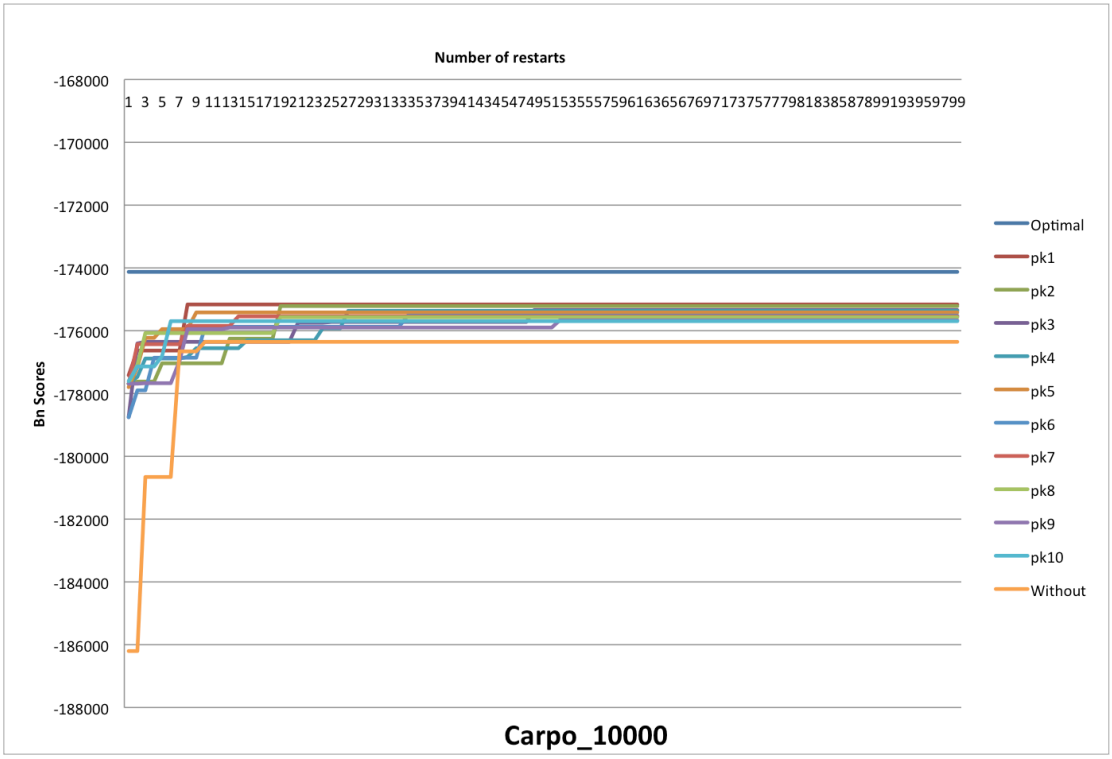

Figure 68: The results of applying HCPK, conditional independence, for synthetic data generated by the Carpo 10000.

6.2.3.2 Backtrack approach

Intelligent backtracking often applies when a network structure is inconsistent with a constraint. For any single random restart, an algorithm checks for conditional independence. For example, take $(A \perp B \mid Z)$ as the conditional independence prior knowledge specified by the user, where A is a conditional independent of B given Z. The algorithm uses the d-Separation algorithm to check for conditional independence, and discovers the nodes reachable from A given Z via active trails. If the generated network does not meet the user's prior knowledge of conditional independence, the backtrack method is applied (discussed in section 5.3.4.1, algorithm 3).

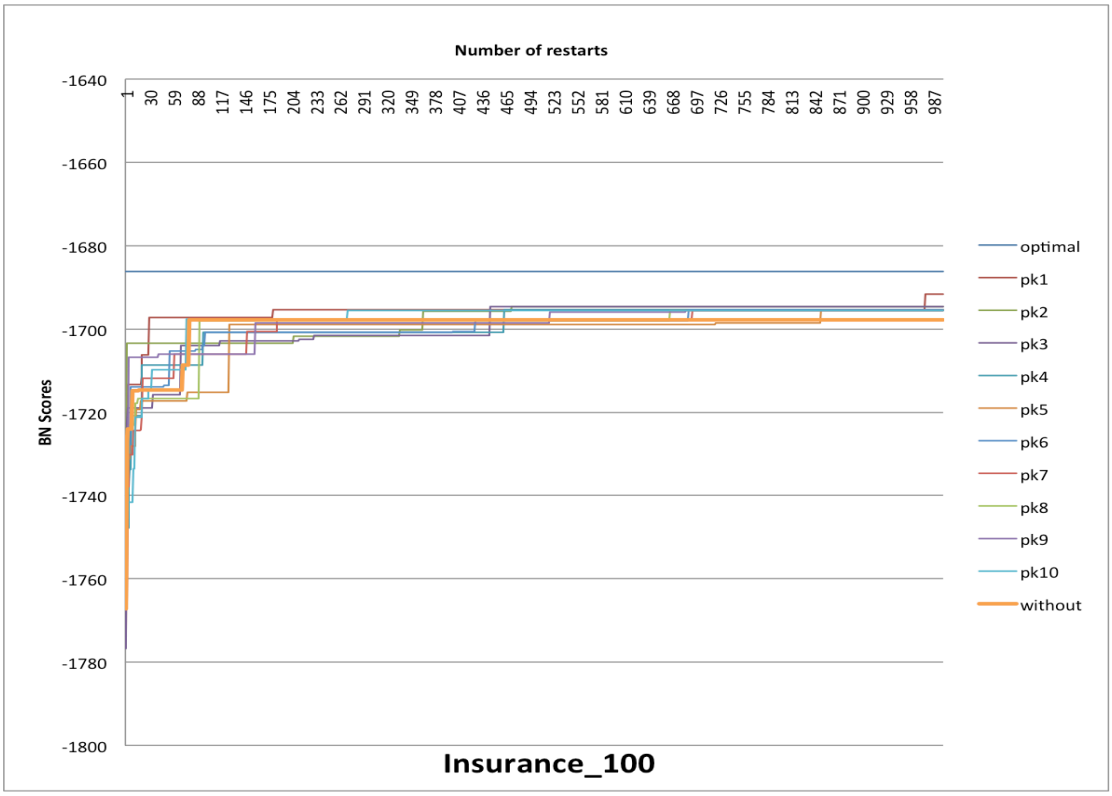

Figure 69: The results of applying HCPK, conditional independence, for synthetic data generated by the Insurance 100.

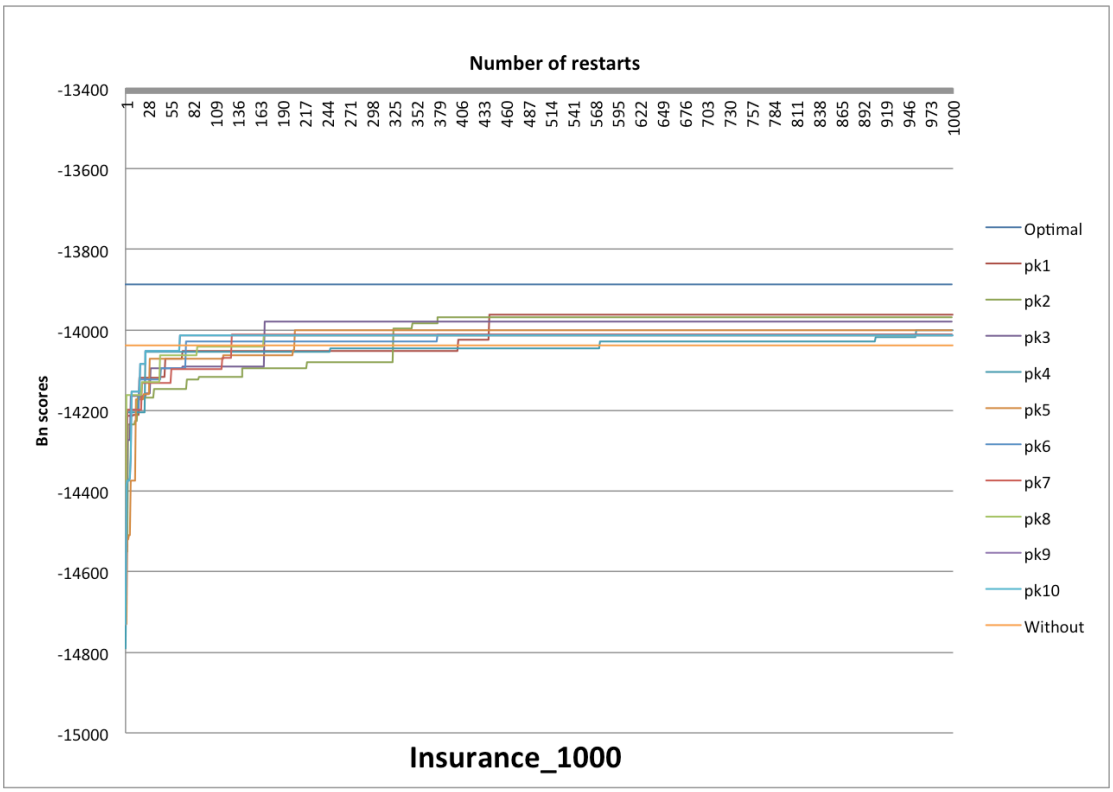

Figure 70: The results of applying HCPK, conditional independence, for synthetic data generated by the Insurance 1000.

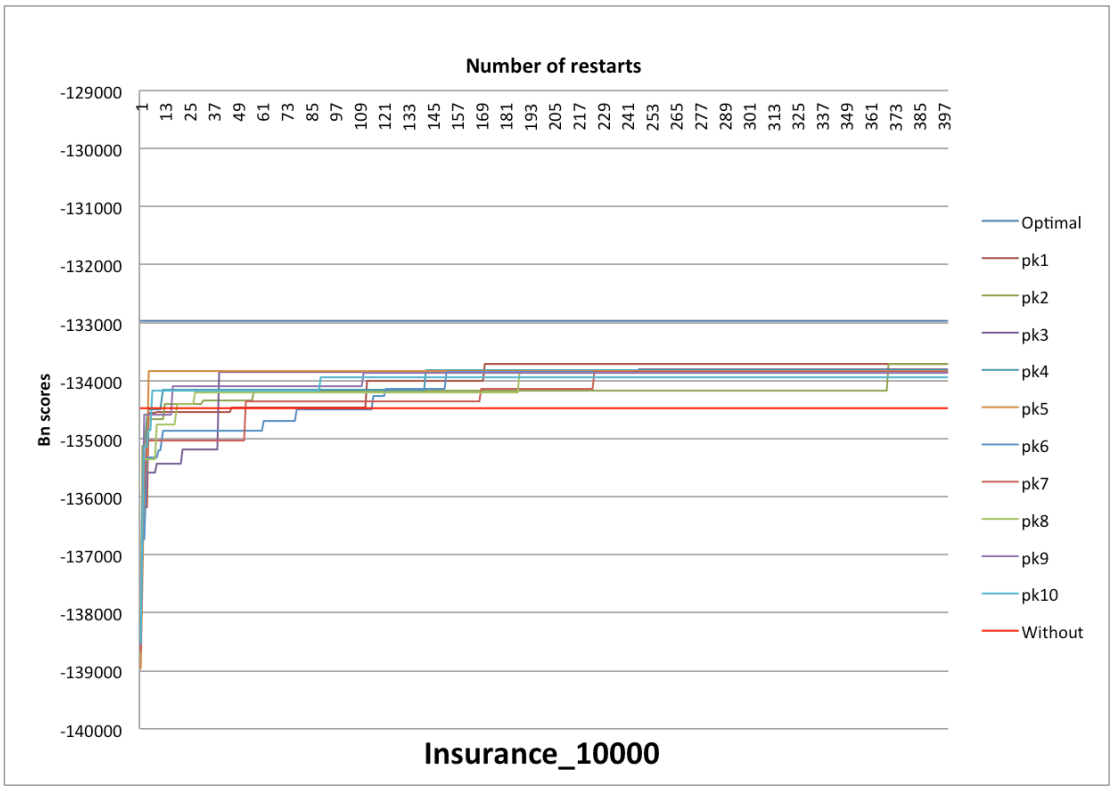

Figure 71: The results of applying HCPK, conditional independence, for synthetic data generated by the Insurance 10000.

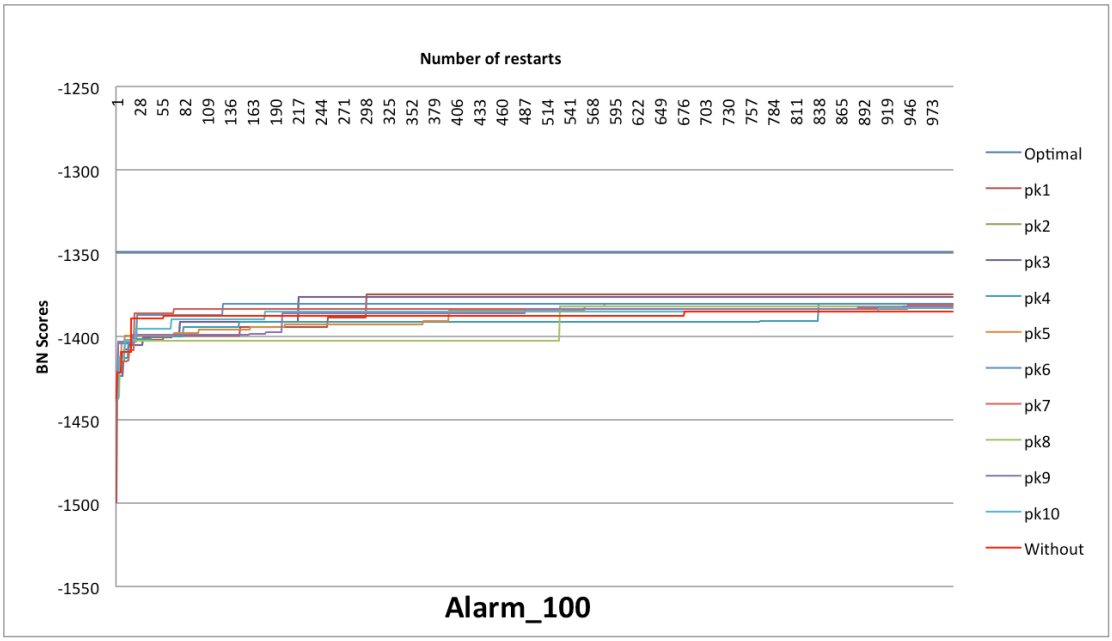

Figure 72: The results of applying HCPK, conditional independence, for synthetic data generated by the Alarm 100.

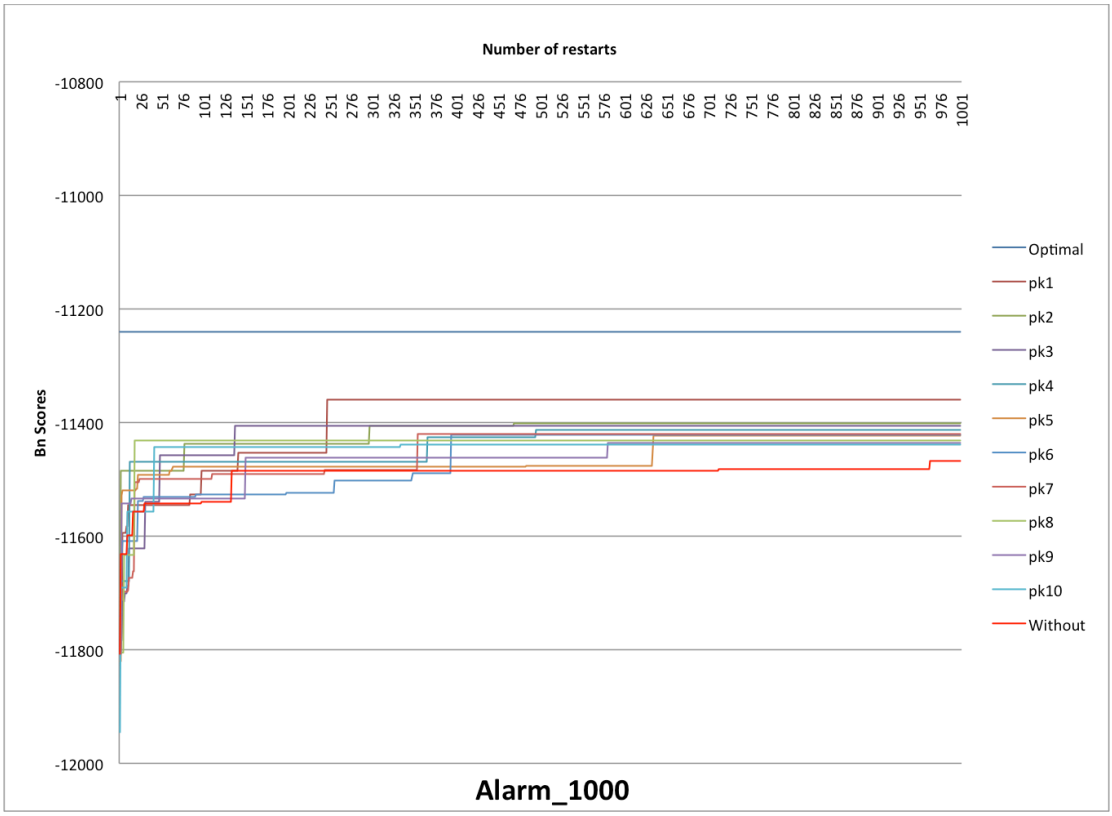

Figure 73: The results of applying HCPK, conditional independence, for synthetic data generated by the Alarm 1000.

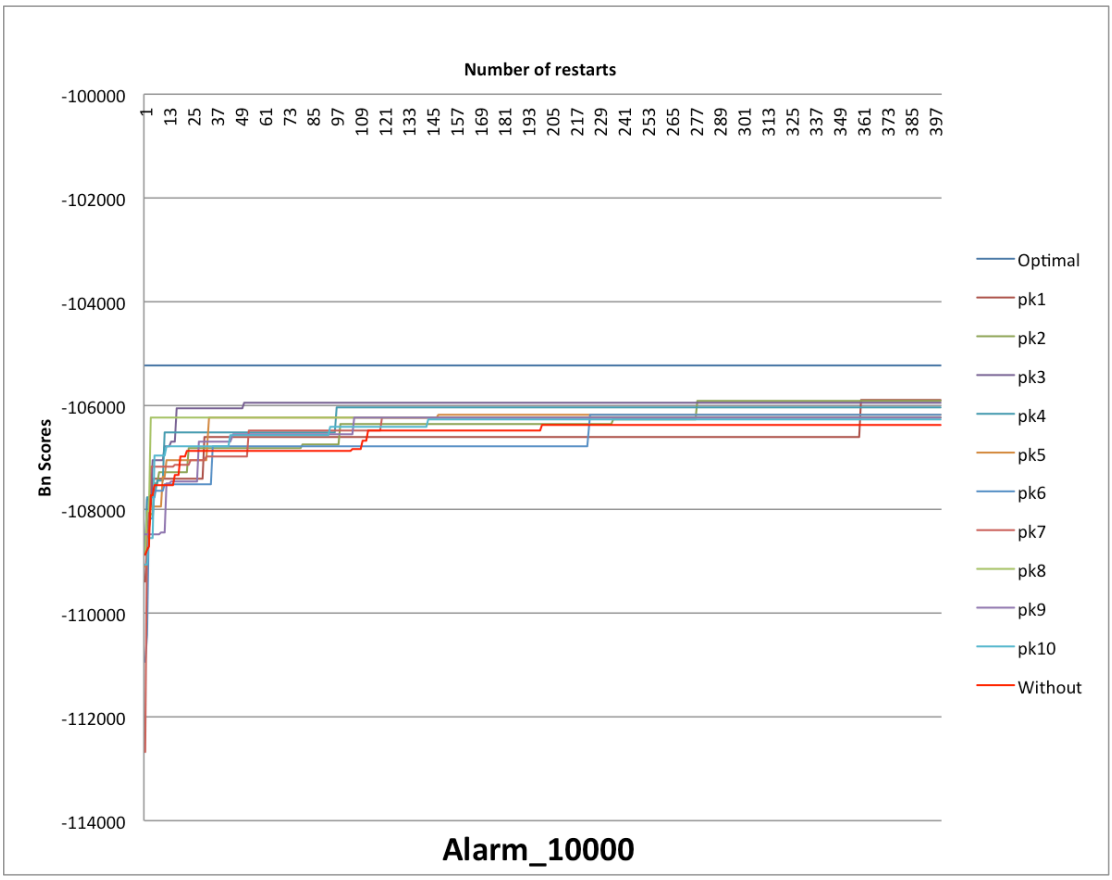

Figure 74: The results of applying HCPK, conditional independence, for synthetic data generated by the Alarm 10000.

Figure 75: The results of applying HCPK, conditional independence, for synthetic data generated by the Mildew 100.

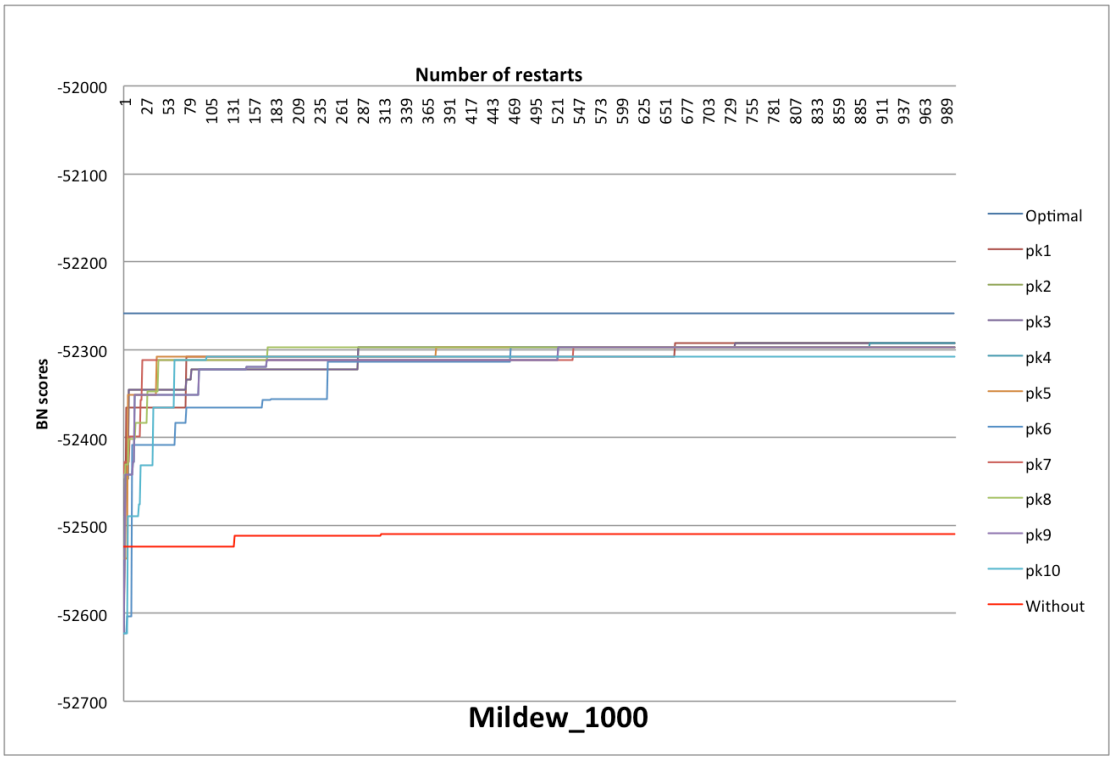

Figure 76: The results of applying HCPK, conditional independence, for synthetic data generated by the Mildew 1000.

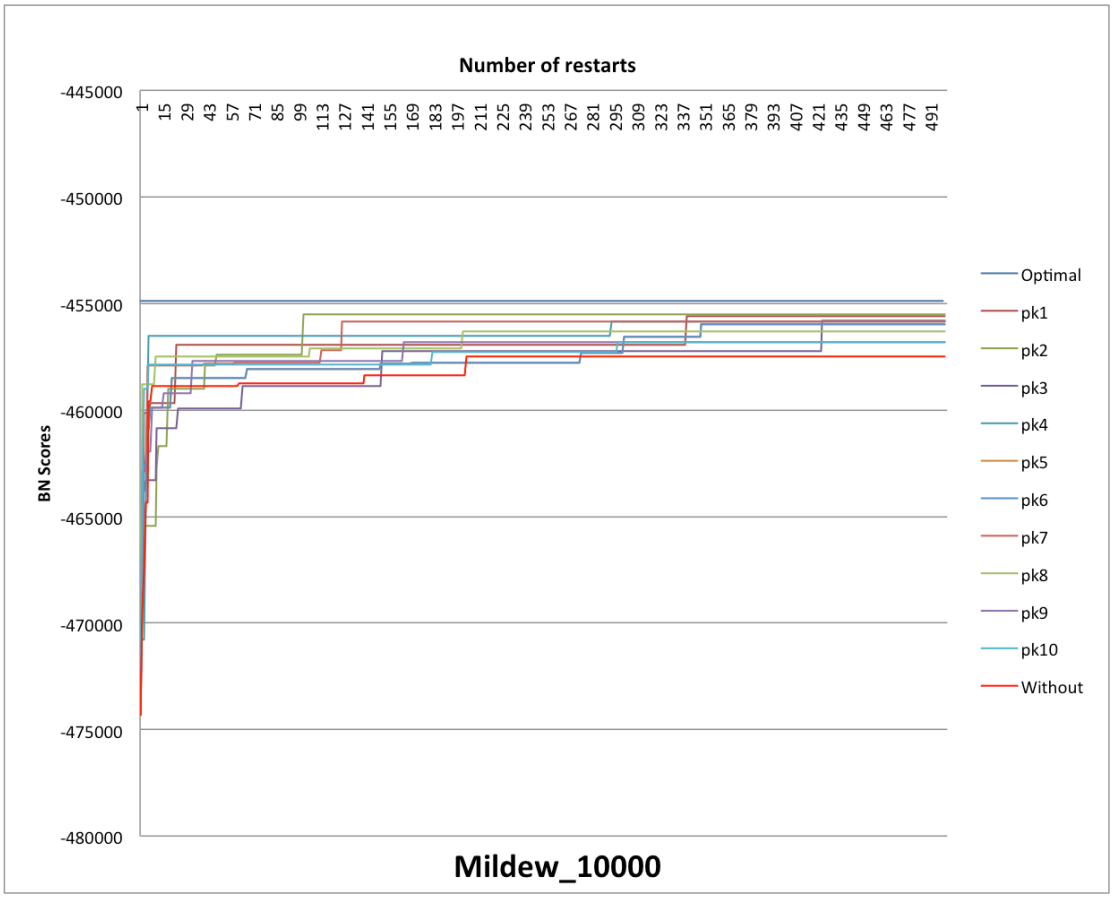

Figure 77: The results of applying HCPK, conditional independence, for synthetic data generated by the Mildew 10000.

The results of incorporating conditional independence prior knowledge, using the conditional independence checks approach, are presented in Figures from 55 to 66. The results of incorporating conditional independence prior knowledge, using the backtrack approach, are presented in Figures from 67 to 75. These results show that the effects of prior knowledge when users specify conditional independence knowledge, using a backtrack approach and a conditional independence checks approach, typically have a positive effect on the learning (Please see Appendix-D for further details).

However, using a dataset where the local scores tend to be quite similar to each other, such as mildew datasets for size 100, demonstrates that the conditional independence prior knowledge has less effect on the learning in both approaches.
For larger dataset sizes, the conditional independence prior knowledge also has a positive effect on the learning in both approaches.

6.2.4 Inconsistent Prior Knowledge

In this research, GOBNILP was used to find the optimal networks, and all prior knowledge was consistent with the optimal network. It has demonstrated that consistent prior knowledge typically has a positive beneficial effect on the learning algorithm, but inconsistent prior knowledge has quite a negative effect on the learning.

The result shows that if we incorporate inconsistent prior knowledge, we get a consistently worse score. Each time we incorporate inconsistent prior knowledge we make the search space smaller, and it becomes more difficult for HCPK to find a high-score network because the really good high-score networks are ruled out.

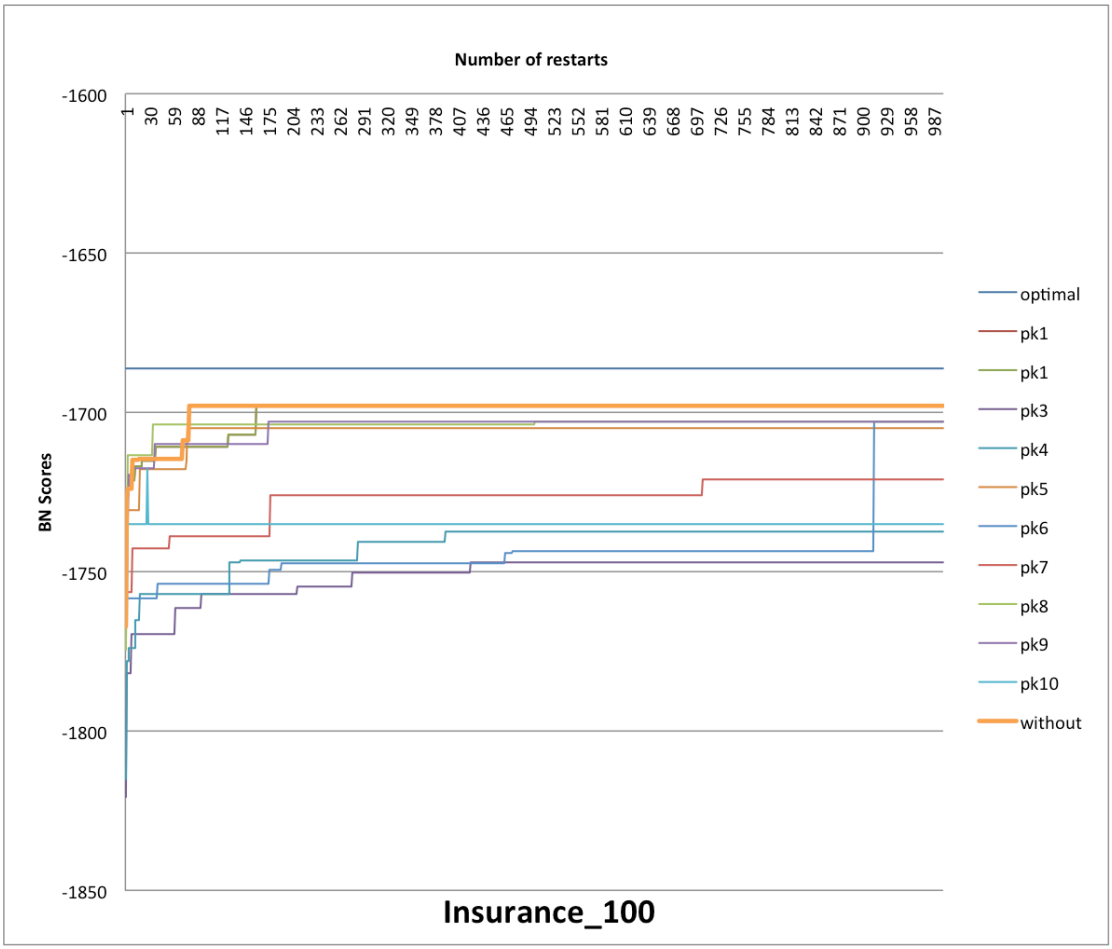

Figure 78: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Insurance 100.

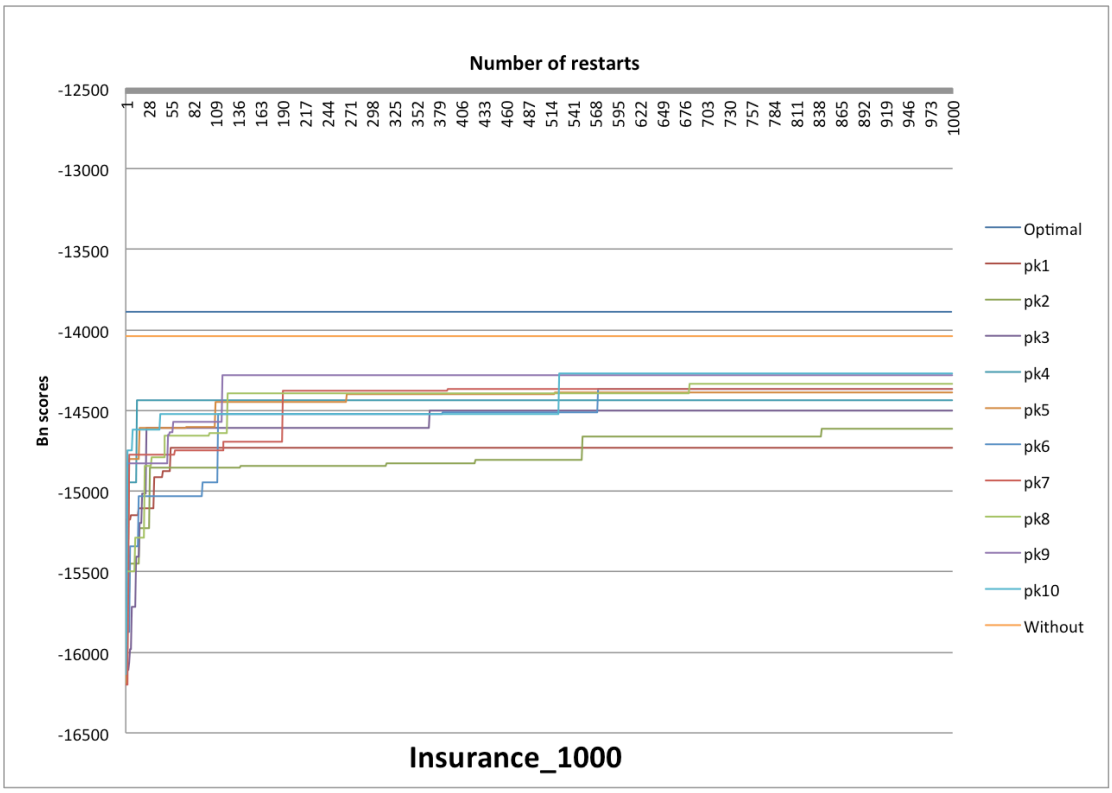

Figure 79: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Insurance 1000.

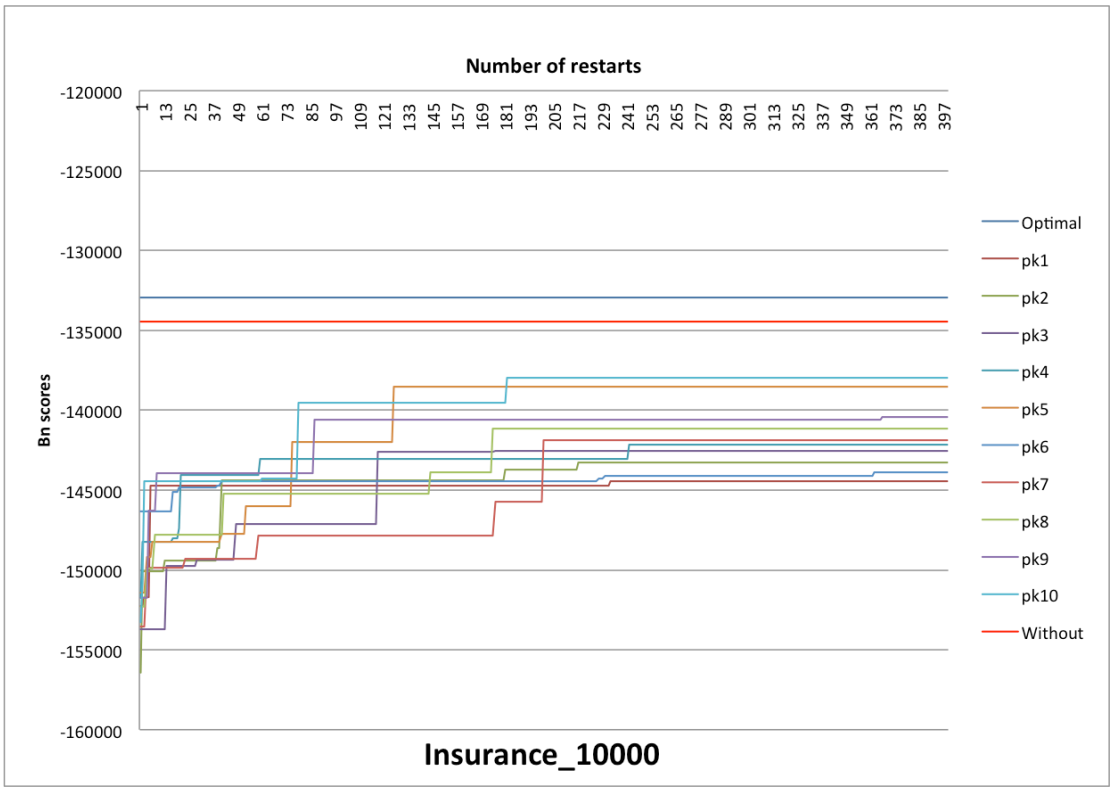

Figure 80: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Insurance 10000.

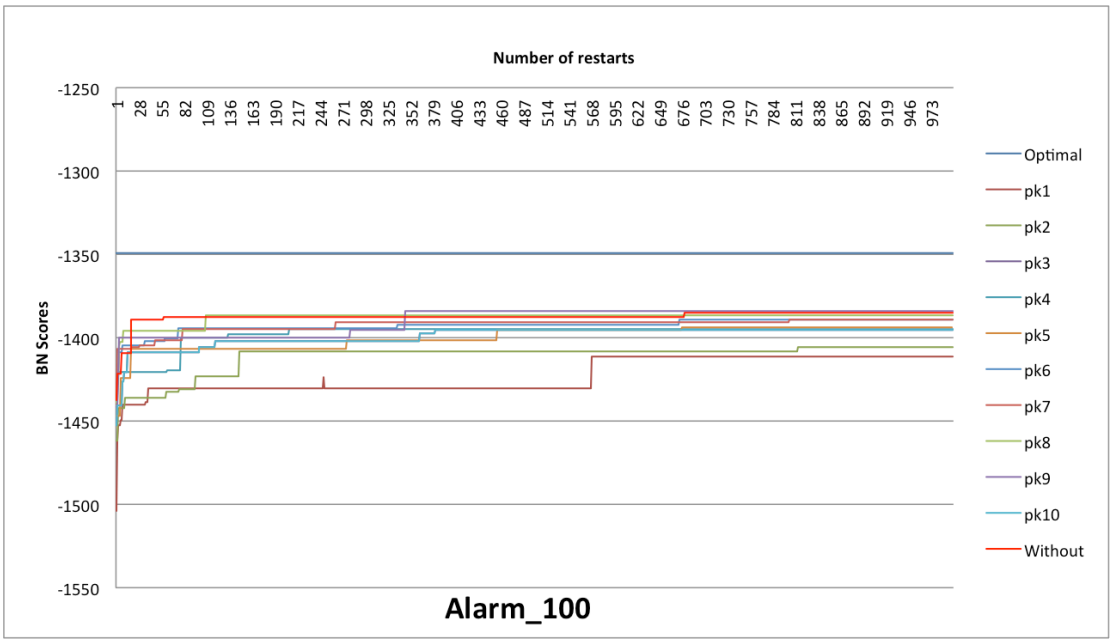

Figure 81: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Alarm 100.

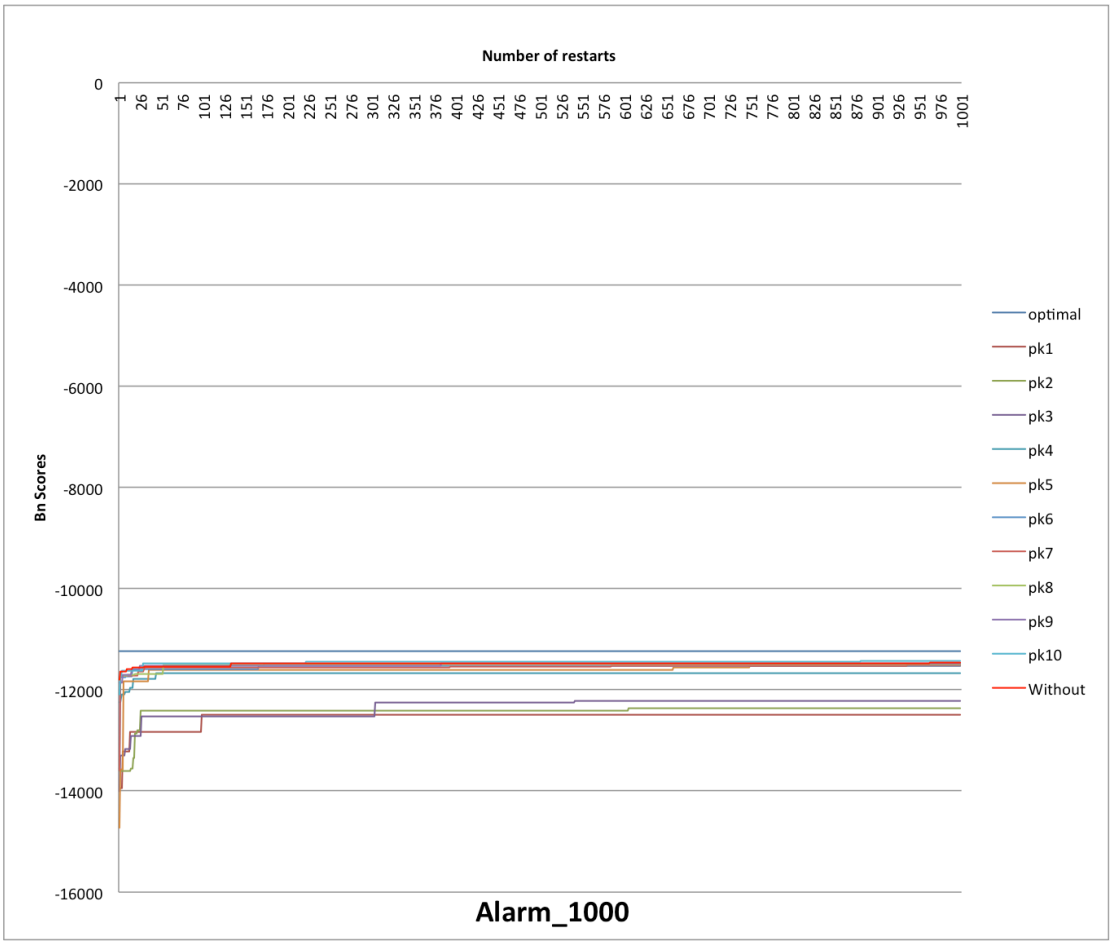

Figure 82: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Alarm 1000.

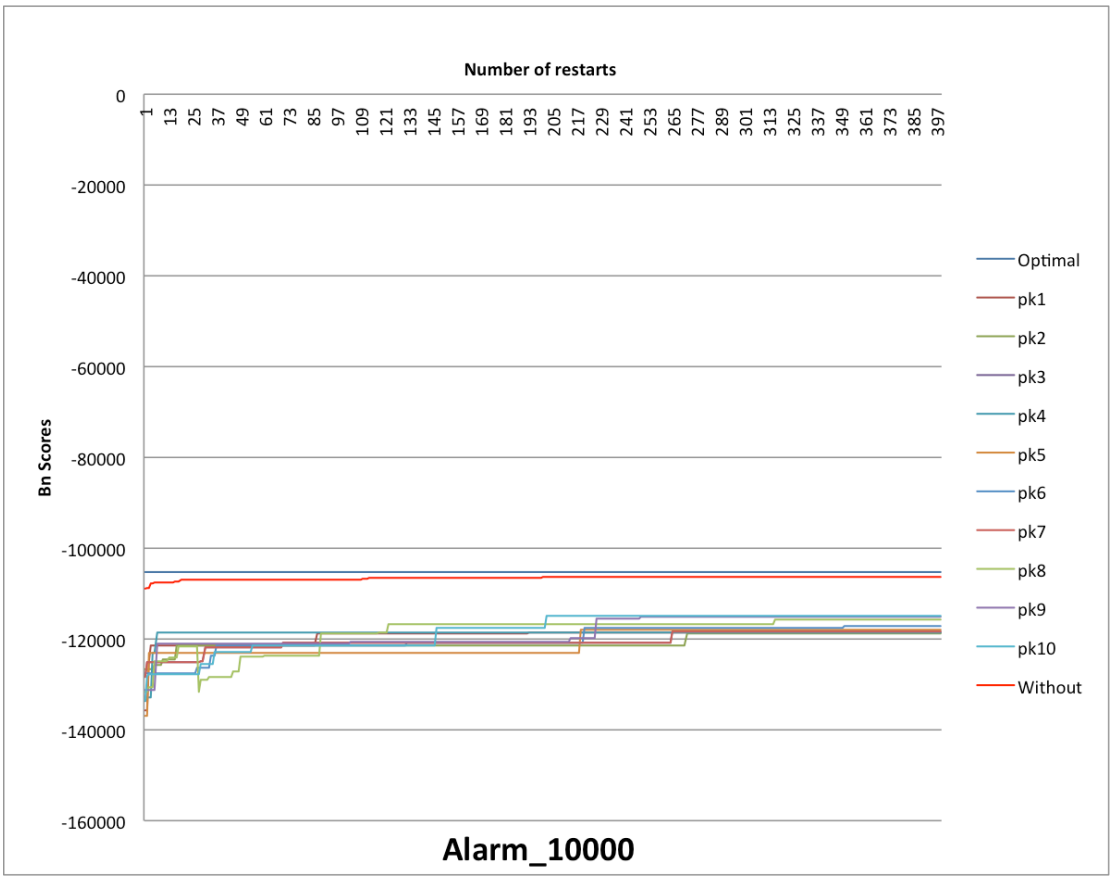

Figure 83: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Alarm 10000.

Figure 84: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Mildew 100.

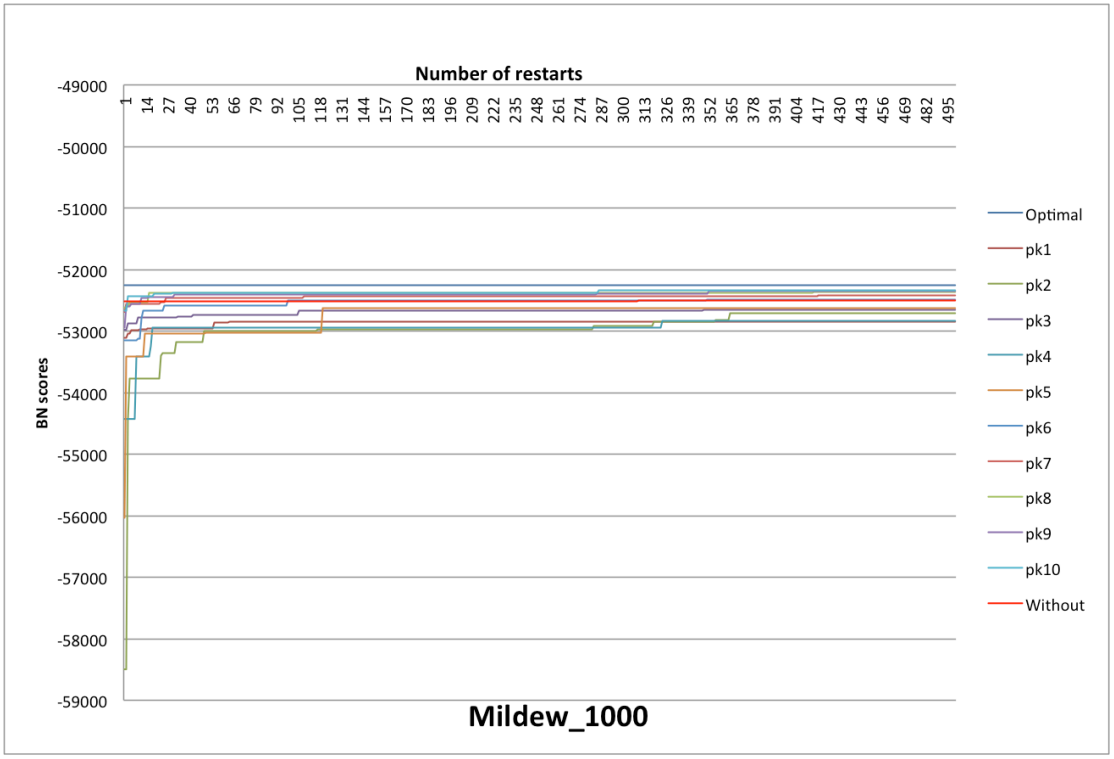

Figure 85: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Mildew 1000.

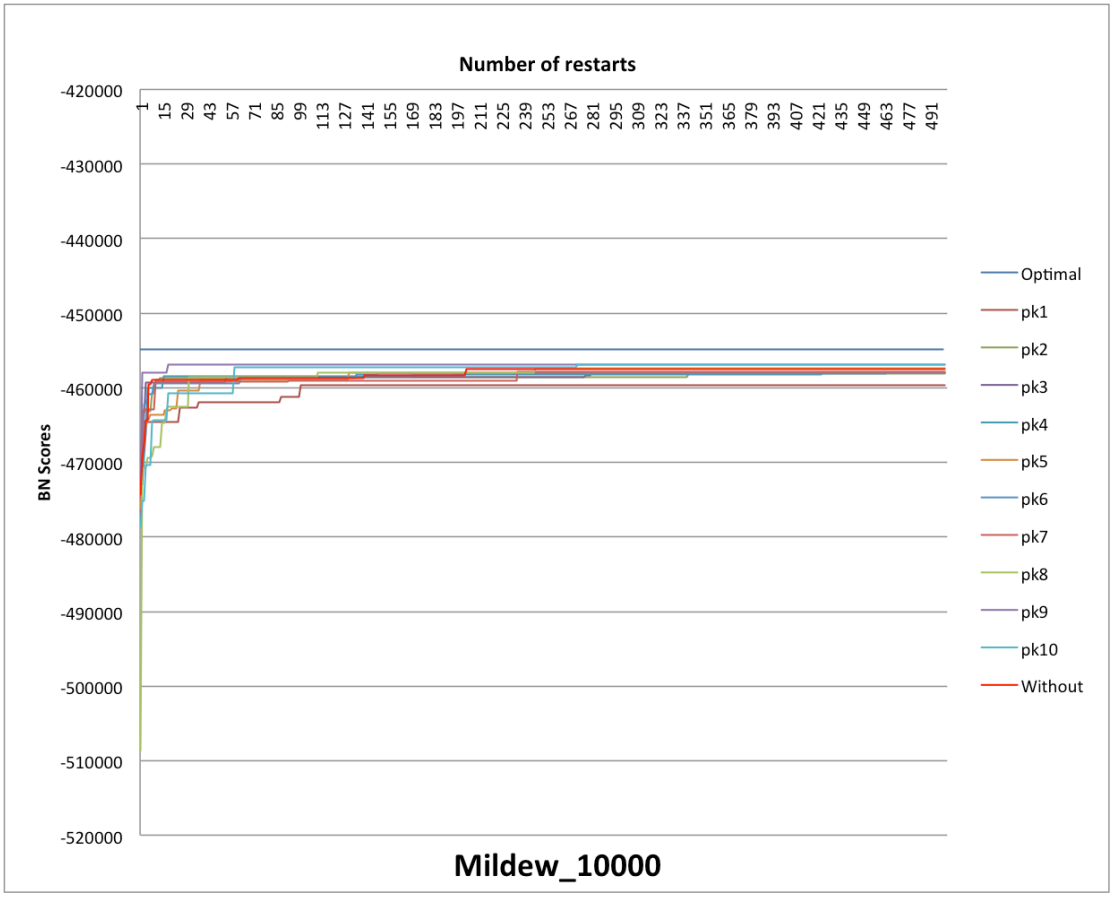

Figure 86: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Mildew 10000.

6.2.5 Learning Bigger problems

In section 6.2, we compared our current algorithm to other existing applications, running three freely available programs on nine different datasets. Table 1 shows the comparison between different applications such as Banjo, GOBNILP and Bnlearn, to HCPK.
However, for bigger problems such as Diabetes and Pigs datasets, the applications execution has stopped that, mainly because the maximum number of states that a variable can assume is limited to 7, or it runs out of available memory. While, our current algorithm without prior knowledge for bigger problems has been solved, for many cases, the simple HCPK gets quite close to the optimal network.

Also, although some applications can find optimal networks on these small examples, it will have problems due to the fact that, for example nodes can have many parents' sets. However, HCPK does not have this problem and can incorporate different sorts of prior knowledge and more complicated knowledge, such as ancestor information and conditional independence, which cannot be incorporated into local scores, thereby leading to a much better network, as it shown in Figures 85 to 90.

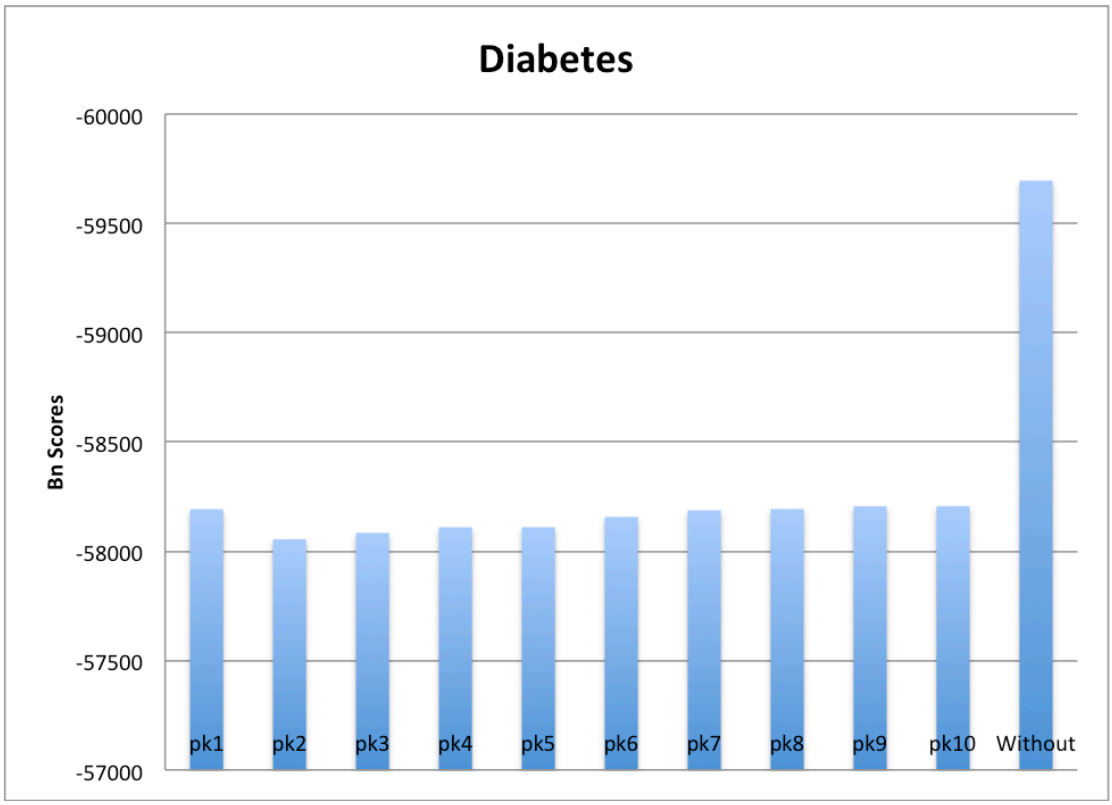

Figure 87: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Diabetes.

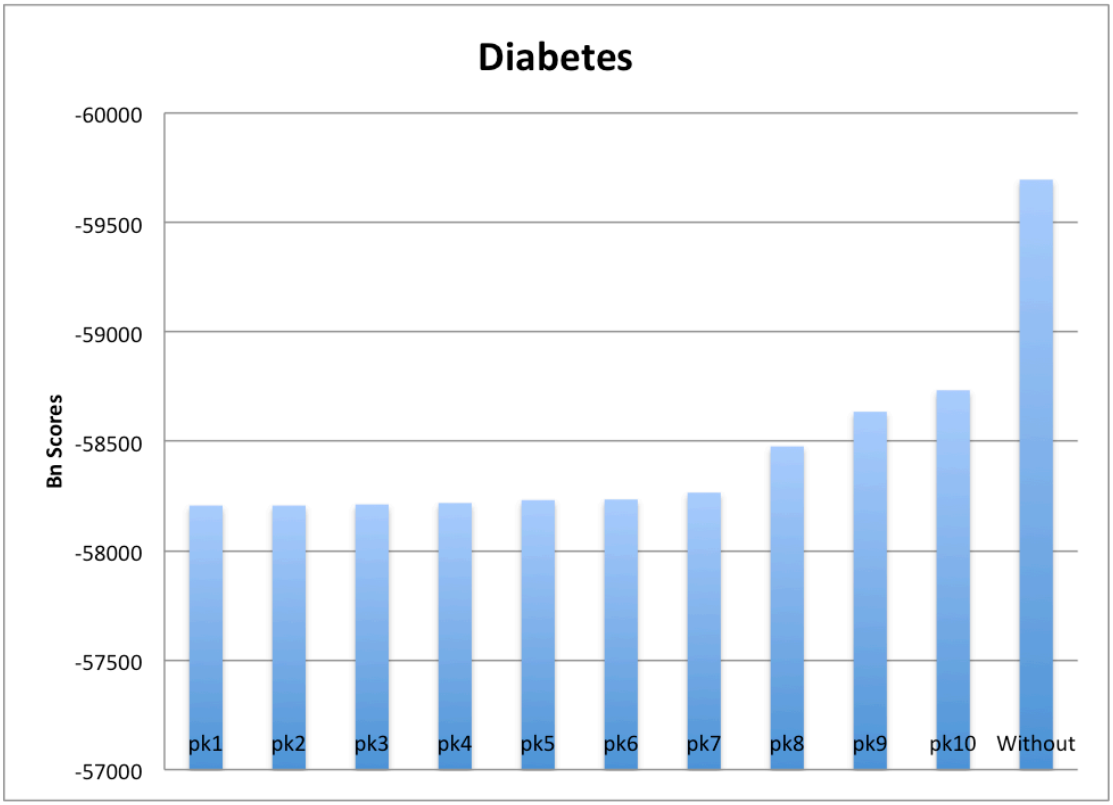

Figure 88: The results of applying HCPK, ancestor relation, for synthetic data generated by the Diabetes.

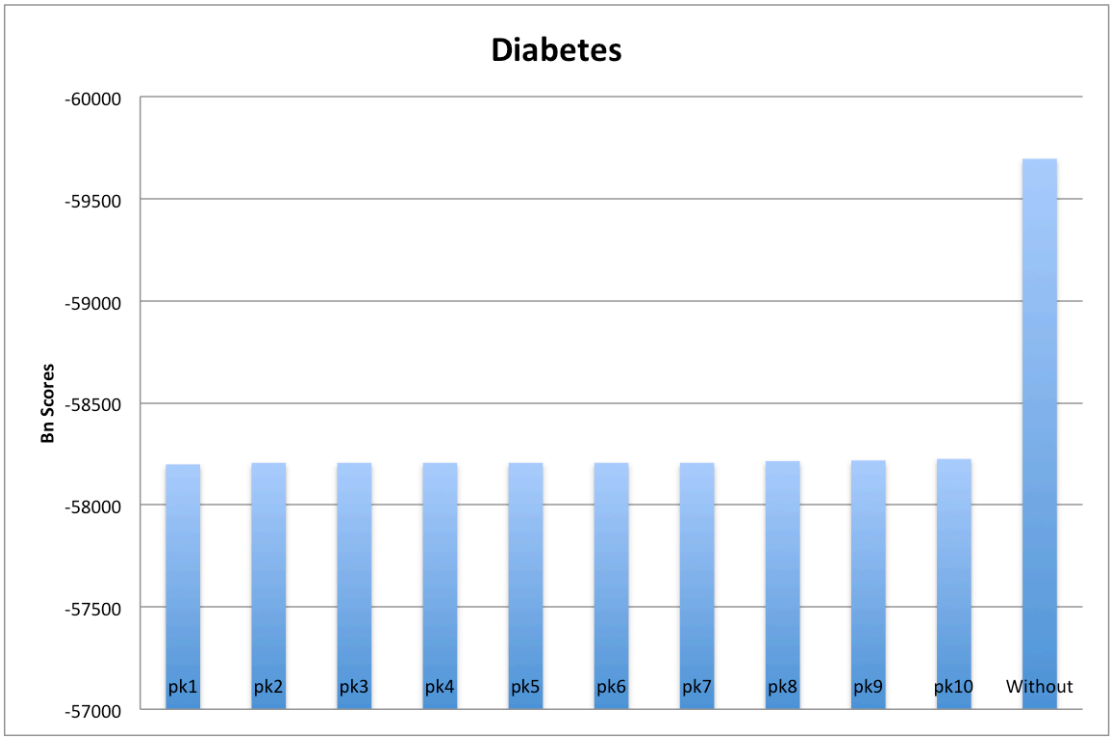

Figure 89: The results of applying HCPK, conditional independence checks, for synthetic data generated by the Diabetes.

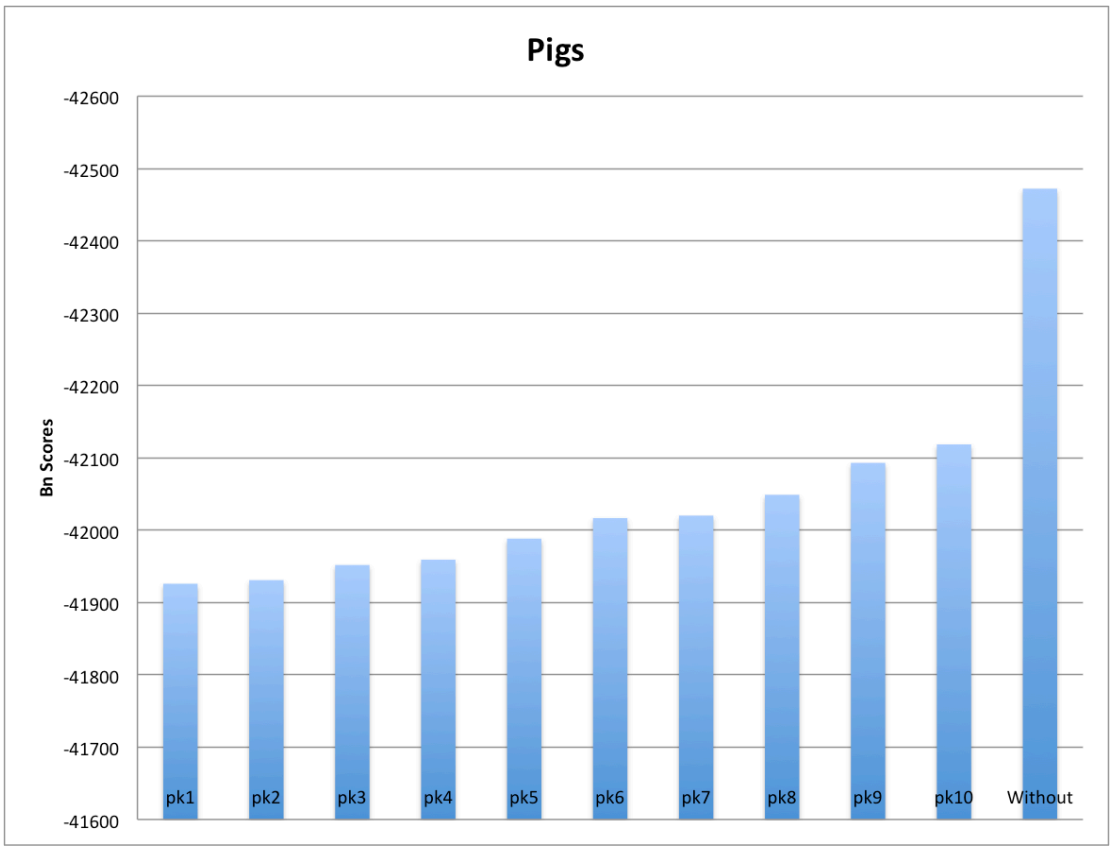

Figure 90: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Pigs.

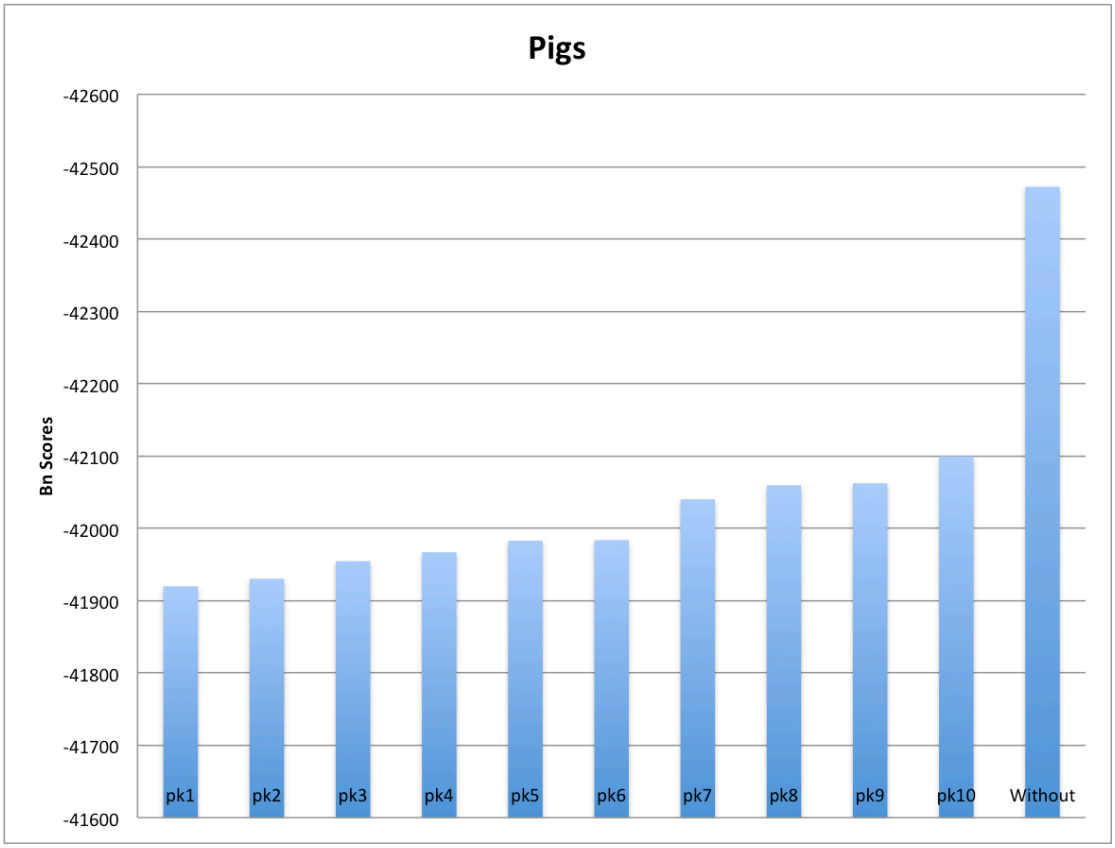

Figure 91: The results of applying HCPK, ancestor relation, for synthetic data generated by the Pigs.

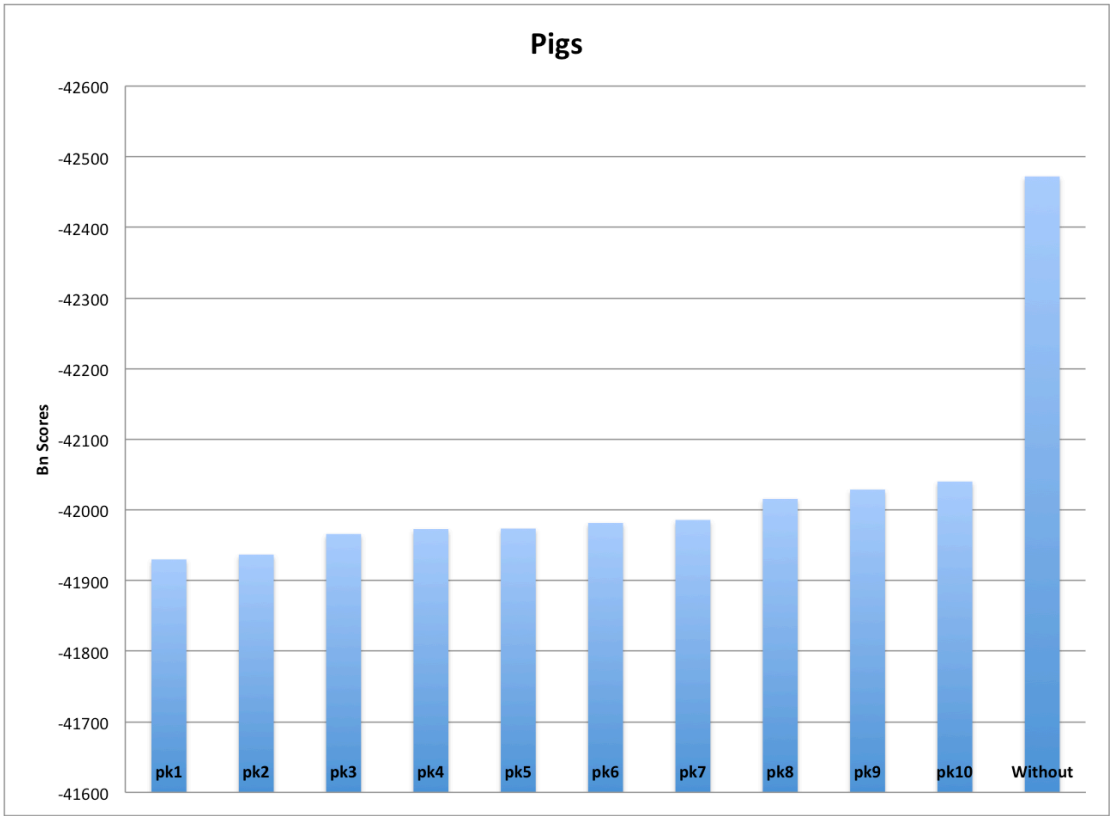

Figure 92: The results of applying HCPK, conditional independence checks, for synthetic data generated by the Pigs.

6.2.1 The execution of HCPK

A summary of the timing results of the execution of HCPK algorithm is in Table 2. We summarise the time of HCPK algorithm, which shown in '100', '1000', and '10000' columns and displays the time in seconds.

We summarise the time of HCPK algorithm using different sorts of prior knowledge, which shown in Tables 3, 4 and 5, and displays the time in seconds.

Table 2: Time of HCPK algorithm

Datasets	Number of variables	100	1000	10000
Asia	8	0	0	2
Insurance	27	1	5	43
Water	32	2	8	68
Mildew	35	2	10	96
Alarm	37	2	12	92
Hailfinder	56	5	30	277
Carpo	60	8	44	359
Diabetes	413	2420	>6 Hours	>6 Hours
Pigs	441	2567	>6 Hours	>6 Hours

Table 3: Time of HCPK algorithm, using prior knowledge, for synthetic data generated by the insurance.

Insurance_100		**Insurance_1000**		**Insurance_10000**	
Pk 1 arrows which have to be there	1	Pk 1 arrows which have to be there	4	Pk 3 arrows which have to be there	40
Pk 2 arrows which have to be there	1	Pk 2 arrows which have to be there	4	Pk 4 arrows which have to be there	40
Pk 3 arrows which have to be there	1	Pk 4 arrows which have to be there	4	Pk 5 arrows which have to be there	40
Pk 4 arrows which have to be there	1	Pk 5 arrows which have to be there	4	Pk 1 arrows which have to be there	41
Pk 5 arrows which have to be there	1	Pk 5 ancestor relation	4	Pk 2 arrows which have to be there	42
Pk1 ancestor relation	1	Pk 3 arrows which have to be there	5	Pk 1 ancestor relation	42
Pk3 ancestor relation	1	Pk 1 ancestor relation	5	Pk 2 ancestor relation	42
Pk1 backtrack approach	1	Pk 2 ancestor relation	5	Pk 4 ancestor relation	42
Pk2 backtrack approach	1	Pk 3 ancestor relation	5	Pk 5 ancestor relation	42
Pk3 backtrack approach	1	Pk 4 ancestor relation	5	Without	43
Pk4 backtrack approach	1	Pk1 backtrack approach	5	Pk1 backtrack approach	46
Pk5 backtrack approach	1	Pk2 backtrack approach	5	Pk 1 conditional independence checks	46
Pk 1 conditional independence checks	1	Pk3 backtrack approach	5	Pk3 backtrack approach	47
Pk2 conditional independence checks	1	Pk4 backtrack approach	5	Pk4 backtrack approach	47
Pk2 conditional independence checks	1	Pk5 backtrack approach	5	Pk5 backtrack approach	47
Pk3conditional independence checks	1	Pk 1 conditional independence checks	5	Pk2 conditional independence checks	47
Pk4 conditional independence checks	1	Pk2 conditional independence checks	5	Pk2 conditional independence checks	47
Pk5 conditional independence checks	1	Pk2 conditional independence checks	5	Pk3conditional independence checks	47
Pk2 ancestor relation	2	Pk3conditional independence checks	5	Pk5 conditional independence checks	47
Pk4 ancestor relation	2	Pk4 conditional independence checks	5	Pk2 backtrack approach	48
Pk5 ancestor relation	2	Pk5 conditional independence checks	5	Pk4 conditional independence checks	48
Without	1	Without	5	Pk 3 ancestor relation	74

Table 4: Time of HCPK algorithm, using prior knowledge, for synthetic data generated by the Mildew.

Mildew _100		**Mildew _1000**		**Mildew _10000**	
Pk 1 arrows which have to be there	2	Pk 5 arrows which have to be there		Pk 3 arrows which have to be there	90
Pk 2 arrows which have to be there	2	Pk 1 arrows which have to be there	9	Pk 5 arrows which have to be there	91
Pk 3 arrows which have to be there	2	Pk 2 arrows which have to be there	9	Pk 4 arrows which have to be there	92
Pk 4 arrows which have to be there	2	Pk 3 arrows which have to be there	9	Pk 1 arrows which have to be there	93
Pk 5 arrows which have to be there	2	Pk 4 arrows which have to be there	9	Pk 2 arrows which have to be there	93
Pk4 ancestor relation	2	Pk3 conditional independence checks	9	Pk 1 conditional independence checks	93
Pk1 backtrack approach	2	Pk4 conditional independence checks	9	Pk3 conditional independence checks	93
Pk2 backtrack approach	2	Pk2 ancestor relation	10	Pk4 conditional independence checks	93
Pk3 backtrack approach	2	Pk3 ancestor relation	10	Pk5 conditional independence checks	93
Pk4 backtrack approach	2	Pk2 backtrack approach	10	Pk1 backtrack approach	94
Pk5 backtrack approach	2	Pk3 backtrack approach	10	Pk2 backtrack approach	94
Pk 1 conditional independence checks	2	Pk4 backtrack approach	10	Pk3 backtrack approach	94
Pk2 conditional independence checks	2	Pk5 backtrack approach	10	Pk4 backtrack approach	94
Pk3 conditional independence checks	2	Pk 1 conditional independence checks	10	Pk1 ancestor relation	96
Pk4 conditional independence checks	2	Pk2 conditional independence checks	10	Pk2 ancestor relation	96
Pk5 conditional independence checks	2	Pk5 conditional independence checks	10	Pk3 ancestor relation	96
Without	2	Without	10	Pk4 ancestor relation	96
Pk3 ancestor relation	6	Pk1 backtrack approach	11	Pk5 backtrack approach	96
Pk1 ancestor relation	7	Pk1 ancestor relation	20	Pk2 conditional independence checks	96
Pk2 ancestor relation	7	Pk4 ancestor relation	21	Without	96
Pk5 ancestor relation	7	Pk5 ancestor relation	22	Pk5 ancestor relation	177

Table 5: Time of HCPK algorithm, using prior knowledge, for synthetic data generated by the Alarm.

Alarm_100		**Alarm_1000**		**Alarm_10000**	
Pk 1 arrows which have to be there	2	Pk 1 arrows which have to be there	11	Pk 1 arrows which have to be there	87
Pk 2 arrows which have to be there	2	Pk 2 arrows which have to be there	11	Pk2 ancestor relation	88
Pk 3 arrows which have to be there	2	Pk 3 arrows which have to be there	11	Pk 3 arrows which have to be there	90
Pk 4 arrows which have to be there	2	Pk 4 arrows which have to be there	11	Pk 5 arrows which have to be there	90
Pk 5 arrows which have to be there	2	Pk 5 arrows which have to be there	11	Pk 2 arrows which have to be there	91
Pk1 ancestor relation	2	Pk4 ancestor relation	11	Pk 4 arrows which have to be there	91
Pk3 ancestor relation	2	Pk5 ancestor relation	11	Without	92
Pk4 ancestor relation	2	Pk1 backtrack approach	12	Pk1 ancestor relation	93
Pk1 backtrack approach	2	Pk2 backtrack approach	12	Pk3 ancestor relation	93
Pk2 backtrack approach	2	Pk3 backtrack approach	12	Pk3 backtrack approach	100
Pk3 backtrack approach	2	Pk4 backtrack approach	12	Pk4 backtrack approach	100
Pk4 backtrack approach	2	Without	12	Pk5 backtrack approach	100
Pk5 backtrack approach	2	Pk5 backtrack approach	13	Pk 1 conditional independence checks	100
Pk 1 conditional independence checks	2	Pk 1 conditional independence checks	13	Pk1 backtrack approach	101
Pk2 conditional independence checks	2	Pk2 conditional independence checks	13	Pk2 backtrack approach	101
Pk3 conditional independence checks	2	Pk3 conditional independence checks	13	Pk2 conditional independence checks	103
Pk4 conditional independence checks	2	Pk4 conditional independence checks	13	Pk3 conditional independence checks	103
Pk5 conditional independence checks	2	Pk5 conditional independence checks	13	Pk5 conditional independence checks	103
Without	3	Pk1 ancestor relation	19	Pk4 conditional independence checks	104
Pk2 ancestor relation	4	Pk3 ancestor relation	20	Pk4 ancestor relation	180
Pk5 ancestor relation	4	Pk2 ancestor relation	26	Pk5 ancestor relation	190

When users specify the parent set, prior knowledge typically has a positive effect on the learning by achieving a high-scoring network and speeds up the learning process. For example, recall the timing execution of HC without prior knowledge in Table 1 for Insurance 1000 dataset, it is 5 seconds. In contrast, when the user specifies the parent sets with prior knowledge 21←2,12 for Insurance 1000 dataset, the timing results of the execution of HCPK is 4 seconds.

In the backtrack approach for a single restart, we backtrack until the user's prior knowledge is satisfied. HCPK constraints are given to generate a network that meets this prior knowledge (as discussed in section 5.3.3.2). Therefore, sometimes HCPK does not need to backtrack as it generates a network that meets this prior knowledge. However, if backtracking is preformed, then the timing of the execution can sometimes take longer than without prior knowledge, as we need to backtrack to a node in the search tree.

For example, when the user specifies the ancestor relation prior knowledge 11⇠22 for Insurance 1000 dataset, the timing results of the execution of HCPK algorithm using the backtrack approach is 8 seconds. Although the timing of the execution can sometimes take longer than without prior knowledge (5 seconds), it satisfies the user's prior knowledge. Whilst it is not difficult to check whether a particular ancestor relation is there, it is difficult to ensure that the graph we are building will satisfy a given ancestor relation. Ancestor relation prior knowledge typically has a positive effect on learning a Bayesian network.

In the conditional independence checks approach for a signal restart, we continue to build the graph by conducting these early checks for conditional independence and, eventually, end up with a network that satisfies the user's prior knowledge. While using the backtrack approach for a signal restart, we backtrack until the user's prior knowledge is satisfied. For example, when the user specifies the conditional independence prior knowledge $18 \perp 16 | 15$ for Insurance 1000 dataset, the timing results of the execution of HCPK algorithm, using the conditional independence checks approach, is 5 seconds, whereas, the timing results of the execution of the HCPK algorithm, using the backtrack approach, is 8 seconds.

Therefore, the conditional independence checks approach achieved faster performance than the backtrack approach so in bigger problems in this thesis, we used the conditional independence checks approach.

6.2.2 True network

In this section we compare the structure of the generated network by HCPK and the true networks. We measure the edges' differences between the true network and the learned network using the structural hamming distance (SHD) algorithm. Moreover, it computes the SHD between two network structures, which is the minimal number of edge additions, edge deletions, and edges reversals required to convert one network structure to another network structure. In this experiment, we used SHD from Bnlearn to compare two different Bayesian networks. In this approach each DAG is converted to a partially directed graph (PDAG), which represents the Markov equivalence class for the DAG. The SHD is then computed on the PDAGs not the original DAGs. Tsamardinos defined SHD as, 'Algorithms that return a DAG are converted to the corresponding PDAG before calculating this measure' (Tsamardinos et al., 2006). See algorithm 6 for computing the SHD.

.

Algorithm 6: SHD Algorithm

```
1.  SHD( Learned PDAG H, True PDAG  G)
2.  Shd = 0
3.  for every edge E diferent in H than G Do
4.          if E is missing in H then
5.              shd+= 1
6.              end if
7.          if E is extra in H then
8.              shd+= 1
9.              end if
10.         if E is incorrectly oriented in H then
11.             shd+= 1
12.             end if
13. end for
```

Tables 6, 7 and 8 shows the comparison between the structure of the generated network by HCPK and the true networks. The symbol + indicates that prior knowledge is consistent with the true network, While symbol * indicates that prior knowledge is inconsistent with the true network but consistent with the optimal network.

Table 6: The comparison between the structure of the generated network by HCPK and the true networks synthetic data generated by the Insurance.

Insurance_100		**Insurance_1000**		**Insurance_10000**	
+19⇠17	35	Optimal	30	Optimal	12
*1←24	36	+23⇠4	33	+4 ⊥ 5 | ∅	28
+23←11,22	36	+13⇠0	33	+13←12,2	31
*4←7	36	+21←2,12	34	+14←4,6	32
+13←2,12	37	*11⇢22	36	+11 ⊥ 21 | 1	33
+23⇠6	37	+23←11,22	37	+10 ⊥ {4,6} | 9	33
+8⇠1	37	+22 ⊥ 20| 1	37	+ 23←11,22	35
+18⇠15	38	+3⊥ 26| 15	37	+26←15	36
+17←7,16	39	+16←11,15	38	+17←7,10,16	37
*24←15	39	+ 16⇠15	38	*10 ⇠ 9	37
*0←3,25	39	*1⊥ {2,12}| 4	39	+21⇠0	39
+14←4,6	39	+6←1	39	* 0←1,25	40
+7⇠6	39	+14←4,6	40	+7⇠1	40
*5⊥7|26	39	+13←2,12	40	*24⇠5	40
*13 ⊥ 21 |12	39	*0⇠4	41	+24←15,23	41
*1 ⊥ 11 | 6	39	*5←7	43	*25 ⊥ 5 | 6	41
Optimal	42	*18⊥ 16| 15	44	*12 ⊥ 15| 13	41
+ 5 ⊥ 10 | ∅	45	*14⊥ 16| 10	44	+8←1,2	42
*11 ⊥ 14 | 6	46	+9←1,2	45	+5 ⊥ 6 | ∅	45
*16 ⊥18,20|15	46	Without	45	*26⇠15	46
Without	48	*11←6,22	46	Without	46

Table 7: The comparison between the structure of the generated network by HCPK and the true networks synthetic data generated by the Alarm.

Alarm_100		**Alarm_1000**		**Alarm_10000**	
+15←10	39	Optimal	18	Optimal	2
+ 25 ⊥ 36 | 24	39	+ 0 ⊥ 15 | 10	29	+24←22,23	32
+ 1 ⊥ 3 | 0	40	+32←18	30	+28←17,27	33
+13←9,12,23	41	+8←2,3,7	32	-21←10,11	33
+17←16+	41	+0 ⊥ 7 | 2	32	+ 1←0	36
* 6←7	41	+7←5,6	33	+24←23	36
+13←10	41	+18←10	34	+26←23	36
+{26,6} ⊥ 34 | 23	41	*2←3	34	+{8,7} ⊥ 9 | 3	37
Optimal	42	+17←16	35	+10←3,8	38
*6←5,7,27	42	+10 ⊥ 11 | ∅	36	*9←0,3	41
+31 ⊥ 1 | 0	42	+8 ⊥ {25,14} | 2	36	+10←3	41
* 8 ⊥ 31|7	42	+27←19,26	37	+15←10	46
+33←3,8	43	+0←1	37	+9 ⊥ 1 | 0	46
+36←24	43	+35←19,20	37	+34 ⊥ {24,26} | 23	46
+35←19,20	44	+13←9,12	37	+5 ⊥ 6 | ∅	46
* 22←16	44	+31 ⊥ 24 | 25	37	+17←16	47
+ 33←8	44	+3 ⊥ 7 | ∅	37	+27←19,26	49
+35 ⊥ 24 | 25	44	+3 ⊥ {12,14} | 10	39	+4←5	49
* 26 ⊥ 16 | ⌀	44	+19←18,34	40	+17 ⊥ 15 | ∅	49
* 23←24,26	45	+{2,3} ⊥ 11 | ∅	40	+12←10,11	50
+12←10,20,21	46	+29←8,15	42	+18←10	50
* 2←16,28,	46	*20←3	42	+19 ⊥ 29 | 18	52
*3←0,9	46	+25 ⊥ 23 | 24	42	+0 ⊥ 3 | ∅	54
+25←24	46	+10←7	43	*4←5	56
+{16,14} ⊥ 3 | 9	46	*21←34	43	+28←27	56
*22←24	47	+6 ⊥ 31 | 7	43	Without	57
*29←8	48	*20←9	46	+31←3	57
Without	49	Without	49	+7 ⊥ 4 | 5	57

Table 8: The comparison between the structure of the generated network by HCPK and the true networks synthetic data generated by the Mildew.

Mildew _100		Mildew _1000		Mildew _10000	
+33←23,27,32	42	+33⇠31	32	Optimal	24
+27←23,24,26	43	*0←11	33	+21←20	29
+14←9	44	+3←1,2	33	+13 ← 8, 12	30
+30←13,29	44	+20⇠1	33	+21⇠20	30
+31←14	44	+9⇠1	33	+24⇠20	31
+32←31	44	+1 ⊥ 2 | φ	33	+32←28,31	32
*23 ⊥ {26, 24} | φ	44	*15←17	34	+8←2,4	34
*0⇠11	44	+9←3	34	+20←3,19	35
*3⇠9	44	Optimal	36	+1 ⊥ 25 | 9	35
+34⇠33	44	+23←9	36	*34←14	36
+31⇠14	44	+17⇠15	41	+27⇠9	36
+32⇠31	44	+2 ⊥ 26 | 14	42	*5←7	37
+ 19 ⊥ 15 | 17	44	*30←13,29	43	+30⇠12	38
+14 ⊥ 32 | 31	44	+26←14	43	+{1,3} ⊥ {25,26} | 14	38
*14 ⊥ 21 | 9	44	+8←2,4	43	+28 ⊥ 31 | φ	38
Without	44	+13 ⊥ 29 | φ	43	+9 ← 3, 8	39
*4←8,27,34	46	+1 ⊥ 20 | 3	43	+30 ← 13, 29	39
*4←8,27,34	46	Without	44	Without	39
+ 1 ⊥ 6 | φ	46	*4⇠31	44	+8 ⊥ 12 | φ	39
+ 22 ⊥ 3 | 9	46	*8⇠13	45	+33⇠14	40
*2←20,26,33	47	*{3,9} ⊥ 22 | 23	45	*0 ⊥ 10 | 11	40
+33⇠32	49	*27←14	46	*16 ⊥ 15 | 17	41
Optimal	63	*0 ⊥ 10 | 11	46	*10⇠11	47

We measure the edges' difference between the true network and the learned network using the SHD algorithm. The optimal network and the true network are quite close, but not quite the same, which helps our algorithm to get close to the true network. However, it is not surprising that, as the amount of data increases, the optimal network is closer to the true network. We learn the Bayesian network model from the data, and, as we have more data, there is a better chance of finding the true network. Moreover, if we do not have much data, it is challenging to learn what the true network is. Usually, adding prior knowledge gives us slightly fewer arrows. Occasionally, the algorithm performs worse without prior knowledge. Interestingly, with bigger datasets, we get bad results without prior knowledge but better ones with prior knowledge. Such consistent prior knowledge, working with the optimal network, which itself is quite close to, but not the same as, the true network, also achieves a better result. For the dataset 100, adding prior knowledge typically has a greater effect on the learning algorithm, as it gets closer to the true network than the optimal.

Generally, when adding prior knowledge, either it is consistent with the optimal network or the true network helps the algorithm to generate a more accurate network. Circumstances in which the number of data points has increased have become more prevalent.

In terms of GOBNILP, when the data are not too large, GOBNILP finds the optimal network, but for bigger datasets, it will fail. For large problems, learning the Bayesian network model from data is NP-hard. We know that, if the problem is too large, there is difficulty in finding the optimal network. Therefore, we had to look for an alternative approach for big cases, and hill-climbing is a good place to start from. Although on these particular datasets, GOBNILP finds the optimal networks, there are other datasets where it cannot work. Also, in previous cases, where GOBNILP limited the parents to three, it worked well, as the true network had three parents as well. However, if we have a problem where variables have, for example 10 parents, GOBNILP will not find them. Adding prior knowledge to HCPK is helpful, but it still cannot find an accurate network as GOPNILP, and this is primarily because of the greedy nature of the hill-climbing approach. Additionally, there is not enough data to identify the true network.

As learning Bayesian networks is NP-hard and these exact learning approaches will not scale to bigger datasets, GOBNILP is not the answer to all problems because of scalability issues. Thus, we have to use an approximate approach and a greedy approach such as hill climbing. Consequently, we need to explore improvements to hill climbing. Our current HCPK deals with bigger problems.

In these experiments, we measure the edges' difference between the true network and the learned network, using the SHD algorithm for 1,000 restarts. For example, in Table 6 for Insurance 100 datasets with prior knowledge 23←11,22, the Bayesian network score is -1700.83, and for the first restart, the difference between the true network and the learned network is 53. After 10 restarts, a better score was found -1700.83, with the difference here being 46. After 1000 restarts, the Bayesian network score was -1691.65 and the difference between the true network and the learned network is 36. Finding the optimal solution thus becomes a question of using enough iteration on the data as we get close to the true network.

This chapter reviewed the research thesis objectives and results. The contributions that this research has made were emphasised, and experiments conducted using the developed algorithm on dynamic programming and HCPK were presented, with and without prior knowledge, while future work is given in the following chapter.

7 Conclusions and future work

This chapter summarises the content of this thesis and highlights its main contributions. It also points out the different directions that could be followed in order to extend these contributions.

The thesis started by providing the necessary background for graphical models. Bayesian networks and Markov networks were then explained, and the Bayesian estimation of a probabilities approach was also described. Thereafter, Bayesian structure learning was then introduced. Firstly, the constraint-based, score-based, and Bayesian model averaging approaches were explained. Then, the thesis went into detail about Bayesian structure learning approaches and applications, following this with approaches that incorporate prior information. The theoretical limits of learning Bayesian networks were discussed.

The thesis proposed algorithms for Bayesian network structure learning, which is described as an optimisation problem. A search-and-score approach was followed, which built upon a hill-climbing algorithm and using the Bayesian Dirichlet likelihood equivalence (BDe) as a scoring metric. This thesis presented a hill-climbing algorithm with prior knowledge (HCPK), a heuristic search, which makes local moves that lead to a locally scored-maximal Bayesian network. The HCPK algorithm is intended to enable users to express their knowledge of a variety of problems in a straightforward manner.

The direction taken by this thesis was the combination of ideas from both the backtrack approach and HC. We demonstrated how to investigate the effect of including ancestor relations prior knowledge via combining the backtrack approach with the HCPK algorithm. In addition to this, we investigated how to develop an approach that builds upon a hill-climbing algorithm and d-Separation algorithm, which was used to investigate the effect of including conditional independence prior knowledge (aside from the backtrack approach). We also showed a comparison of the effects of prior knowledge when users specify conditional independence knowledge,

using the backtrack approach and the developed conditional independence checks approach.

The results of these studies point the way to further directions that could be pursued. For instance, to incorporate conditional independence prior knowledge, we used the backtrack approach. The algorithm uses the d-Separation algorithm to check for conditional independence, and discovers the nodes reachable from given active trails. If the generated network does not meet the user's prior knowledge of conditional independence, the backtrack method is applied. Only conditional independence active trails are considered when the algorithm selects a random variable to backtrack.
A further direction could be taken using the approach suggested by Tian, Paz and Pearl (1998), who noted the problem of finding a minimal separator in Bayesian networks. Therefore, in the future, we should investigate this interesting direction with a view to finding a set of nodes that separates a given pair of nodes such that no suitable subset separates that pair.

In this research, we aimed to keep the hill-climbing as simple as possible to investigate the effect of different sorts of prior knowledge. If we start with an empty graph, then it is possible to construct any Bayesian network by adding edges. Moreover, in this research, we explored the difference between this particular algorithm and the same algorithm with prior knowledge. The local moves in this algorithm are the addition of an edge. Thus, in future work, we will also examine in more detail the results of prior knowledge in a less greedy search approach by adding, deleting and reversing an edge. In addition, another possible direction for future work is adopting a simulated annealing approach, which finds a good solution to an optimization problem where we can avoid getting stuck at local maxima. However, while these solutions tend to be better than any others nearby, they are typically not optimal.

Hill climbing does not have computational space issues as it looks only at the current state. Hill climbing's main source of computational complexity emerges from the time required to explore the problem space. HCPK with Random-restart can, in theory, reach optimal solutions within polynomial time for most problem spaces. However, for some NP-complete problems, the number of local maxima can be the cause of exponential computational time to find an optimal solution. Hence, further investigation needs to be carried out to increase the performance of HCPK, which finds a good solution to an optimization problem.

Appendix A

Table 9: The results of applying the dynamic programming with prior knowledge, arrows which have to be there, for synthetic data generated by the Asia.

Asia_100		**Asia _1000**		**Asia _10000**	
Optimal	-245.64	Optimal	-2317.41	Optimal	-22466.39
Without	-245.64	Without	-2317.41	Without	-22466.39
Prior knowledge consistent with GOBNILP					
0←2	-245.64	0←2	-2317.41	0←2	-22466.39
1←0,4	-245.64	1←0	-2317.41	1←0	-22466.39
3←4	-245.64	4←0,3	-2317.41	3←4	-22466.39
5←1,4	-245.64	4←0	-2317.41	5←1	-22466.39
6←1	-245.64	4←3	-2317.41	6<-5	-22466.39
7←2	-245.64	7←2	-2317.41	7<-2	-22466.39
Inconsistent prior knowledge with GOBNILP					
1←6	-245.65	1←5	-2318.53	5←0	-22472.05
5←6	-245.65	5←6	-2318.53	4←0,5	-22474.39
5←6,4	-245.65	5←0	-2319.45	7←4	-22475.19
1←6	-245.65	4←6	-2319.92	7←1	-22475.19
4←5	-245.65	0←6	-2322.29	5←2	-22476.64
4←1,5	-245.79	6←0	-2322.29	0←6	-22477.27
0←1,7	-247.28	5←2,7	-2324.43	5←2,7	-22477.60
0←6	-247.74	6←7	-2329.05	1←2,5	-22478.55
1←0,6	-247.74	1←7,6	-2333.12	2←6,7	-22479.91
5←0,6	-247.74	0←7	-2335.26	6←7	-22481.51
6←0	-247.74	0←7,5	-2335.26	1←7	-22484.05
7←0	-248.51	2←7,0	-2335.26	0←7	-22495.82

Table 10: The results of applying the dynamic programming with prior knowledge, known ordering, for synthetic data generated by the Asia.

Asia_100		**Asia _1000**		**Asia _10000**	
Optimal	-245.64	Optimal	-2317.41	Optimal	-22466.39
Without	-245.64	Without	-2317.41	Without	-22466.39
Prior knowledge consistent with GOBNILP					
2<0	-245.64	3<5	-2317.41	5<6	-22466.39
0<1	-245.64	2<7	-2317.41	0<5	-22466.39
1<6	-245.64	0<7	-2317.41	2<7	-22466.39
5<6	-245.64	1<5	-2317.41	0<7	-22466.39
2<7	-245.64	3<6	-2317.41	1<6	-22466.39
0<5	-245.64	2<1	-2317.41	2<0	-22466.39
Inconsistent prior knowledge with GOBNILP					
1<4	-245.65	4<0	-2317.66	5<4	22468.53
1<0	-245.65	6<1	-2318.53	5<1	-22468.53
5<0	-245.65	6<5	-2318.53	6<1	-22468.53
6<0	-245.65	6<4	-2318.53	6<5	22468.53
6<4	-245.65	7<5	-2322.58	7<0	-22473.62
5<4	-245.65	7<2	-2323.56	7<5	-22475.19
5<7	-245.64	6<7	-2317.41	6<7	-22466.39
3<2	-245.64	2<5	-2317.41	2<0	-22466.39
6<7	-245.64	4<1	-2317.41	4<3	-22466.39
3<1	-245.64	3<2	-2317.41	0<5	-22466.39

Table 11: The results of applying the dynamic programming with priors knowledge, known ordering and arrows which have to be there, for synthetic data generated by the Kredit.

Kredit 10000			
Prior knowledge consistent with GOBNILP			
Arrows which have to be there	Arrows which have to be there	Ordering	BNs score
0←4	0←12	13<0	-16695.66
1←10,17	1←0	12<1	-16695.66
2←15	2←0	15<9	-16695.66
4←14	4←0	14<11	-16695.66
5←4	5←14	16<17	-16695.66
8←5	8←17	6<10	-16695.66
9←15	9←5	0<12	-16695.66
10←7	10←12	5<14	-16695.66
11←14	11←2	8<4	-16695.66
12←0	12←9	4<1	-16695.66
14←13	14←16	9<5	-16695.66
15←14	15←7	3<5	-16695.66
17←4	17←10,16	11<1	-16695.66
Inconsistent prior knowledge with GOBNILP			
Arrows which have to be there	BNs score	Ordering	BNs score
10←8	-16696.48	15<12	-16695.66
1←10	-16697.11	14<10	-16695.66
17←15	-16697.36	12<15	-16695.66
10←17	-16698.00	10<8	-16695.66
17←12	-16701.41	7<12	-16695.66
16←17	-16701.89	16<17	-16695.66
14←10	-16709.24	11<6	-16695.66
15←12	-16710.25	9<12	-16695.66
12←15	-16710.25	17<16	-16697.36
10←16	-16714.73	17<10	-16698.00

Appendix B

Table 12: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Insurance.

Insurance_100		**Insurance_1000**		**Insurance_10000**	
Optimal	-1686.22	Optimal	-13887.35	Optimal	-132968.57
1←24	-1691.65	*21←2,12	-13966.04	*13←12,2	-133655.01
*17←7,16	-1691.65	*19←17,18	-13972.82	* 23←11,22	-133657.64
24←15	-1691.65	*7←4,6	-13973.29	* 19←17,18	-133732.17
*23←11,22	-1691.65	*14←4,6	-13983.54	*17←7,10,16	133744.11
*13←2,12	-1691.67	23←11,22	-13986.19	*21←2,12	-133825.48
25←14,20	-1693.52	11←6,22	-13987.26	*8←1,2	-133828.22
*0←3,25	-1693.73	*16←11,15	-13992.23	*24←15,23	-133898.14
*14←4,6	-1693.73	*9←1,2	-14002.44	*15←5,13	-133941.81
6←1	-1694.60	2←0,3	-14014.37	6←1	-133959.36
*9←1,2	-1695.34	6←1	-14017.12	*14←4,6	-133962.48
11←6	-1695.42	*8←1,2	-14018.59	*7←4,5,6	-134028.62
*8←1,2	-1695.49	*13←2,12	-14018.95	*16←11,15	-134075.60
4←7	-1695.60	10←4	-14021.98	3←0,2	-134100.02
7←6	-1695.76	*0←3,10	-14022.36	*22←4,6	-134106.98
20←1	-1695.76	5←7	-14022.76	0←1,25	-134156.58
21←12	-1695.78	*22←4,6	-14025.06	26←15	-134182.02
15←13	-1695.88	*17←7,10,16	-14025.66	Without	-134474.33
3←2	-1696.07	26←15	-14033.05	12←0,3	-134490.17
19←17,18	-1696.52	Without	-14038.33	4←1	-140288.41
Without	-1697.81	25←0	-14262.08	25←20,9	-140660.06

Table 13: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Mildew.

Mildew _100		**Mildew _1000**		**Mildew _10000**	
Optimal	-6380.72	Optimal	-52258.85	Optimal	-454894.40
*33←23,27,32	-6543.71	23←9	-52258.85	*13 ← 8, 12	-455746.79
*27←23,24,26	-6586.02	0←11	-52264.28	21←20	-455801.10
*2←20,26,33	-6633.54	15←17	-52291.72	*9 ← 3, 8	-455834.56
*4←8,27,34	-6646.04	10←11	-52292.33	*8←2,4	-455846.98
*12←20,28,33	-6647.73	13←31	-52292.33	*30 ← 13, 29	-455861.47
*11←12,27,29	-6648.03	19←20	-52292.33	*20←3,19	-455987.05
*29←3,27,34	-6648.43	33←34	-52292.33	34←14	-456050.35
*21←22,24,27	-6651.99	27←14	-52292.33	*32←28,31	-456154.28
*17←19,24,34	-6652.01	4←13	-52292.33	0←11	-456167.67
9←21,24,27	-6652.26	*8←2,4	-52297.75	16←17	-456220.68
*7←12,23,27	-6652.51	*30←13,29	-52297.75	5←7	-456243.40
Without	-6658.16	22←23	-52297.75	*24 ← 21, 23	-456285.25
20←21	-6658.16	34←31	-52304.02	*23←9,22	-456320.52
*30←13,29	-6658.16	9←3	-52307.13	27←14	-456491.04
31←14	-6658.16	12←13	-52307.60	*14←9,13	-456632.90
5←7	-6658.16	31←14	-52307.60	*26←14,25	-456656.59
14←9	-6658.16	*3←1,2	-52318.82	*3←4,5	-456879.13
15←17	-6658.16	26←14	-52318.92	Without	-457258.64
32←31	-6658.16	Without	-52510.18	4←8,9	-457951.11

Table 14: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Alarm.

Alarm_100		**Alarm_1000**		**Alarm_10000**	
Optimal	-1349.22	Optimal	-11240.34	Optimal	-105226.51
*13←9,12,23	-1373.17	*27←19,26	-11399.58	*28←17,27	-105850.61
*12←10,20,21	-1376.28	*8←2,3,7	-11413.76	15←10	-105869.75
25←24	-1376.32	20←9	-11414.01	*26←22,23	-105899.20
*33←3,8	-1377.382	*29←8,15	-11416.55	*27←19,26	-105969.99
*2←16,28,	-1378.15	0←1	-11419.20	*21←10,11	-106083.38
3←0,9	-1378.44	*19←18,34	-11422.43	*12←10,11	-106098.35
*0←7,9	-1379.44	36←24	-11423.77	1←0	-106106.34
*35←19,20	-1379.48	*35←19,20	-11427.70	*24←22,23	-106131.71
15←10	-1380.02	*7←5,6	-11427.82	10←3,8	-106152.00
11←20	-1380.31	*13←9,12	-11429.52	34←23	-106167.89
17←16	-1380.54	15←10	-11439.95	*29←8,15	-106190.31
*23←24,26	-1380.71	22←23,24	-11441.57	*9←0,3	-106193.67
21←2,19	-1381.51	*9←0,3	-11447.16	*32←19,30	-106219.26
*4←5,12,22	-1381.96	*21←19,20	-11450.40	*35←19,20	-106300.84
*6←5,7,27	-1382.29	*1←2,11	-11450.66	*8←2,3,7	-106334.40
*10←0,3,8	-1382.51	26←22,23	-11452.63	4←5	-106345.22
*18←0,15,17	-1382.53	10←3,8	-11456.17	*31←2,3,7	-106353.83
*22←16,26,35	-1382.58	*31←3,7	-11465.51	*18←15,17	-106398.84
*20←18,21	-1383.15	*18←15,17	-11466.51	19←18	-106464.83
Without	-1384.68	Without	-11467.32	Without	-106496.20
8←7	-1387.51	23←22,24	-11470.89	33←3,8	-106536.36

Table 15: The results of applying HCPK, arrows which have to be there, for synthetic data generated by the Carpo.

Carpo_100		**Carpo_1000**		**Carpo_10000**	
Optimal	-1829.30	Optimal	-17718.94	Optimal	-174130.56
*11←40,46,52	-1854.83	*27←0,22,24	-17855.22	52← 6	-175364.94
*2←35,49	-1856.71	*16←11,12,15	-17856.72	*16←11,14,15	-175393.50
*10←3,12,26	-1857.71	*47←12,24	-17860.41	*30←0,6,20	-175669.97
*33←20,24	-1857.97	*58←0,22,53	-17871.78	*47←8,24	-175755.71
*40←17,18	-1858.50	*36←15,51,53	-17875.23	*33←6,20,24	-175822.23
4*←13,32,43	-1858.52	40←16	-17876.11	*25←20,24,33	-175826.03
*24←1,37	-1858.81	*49←25,35	-17879.82	*7←6,8	-175830.59
21←20	-1859.48	*4←3,6,13	-17882.18	12←11,16	-175881.79
*38←0,24,59	-1859.70	*50←0,8,59	-17883.44	*20←11,19	-175902.89
*55←2,20,59	-1860.02	*23←9,22	-17885.38	*44←0,20,22	-175912.76
*0←2,24,35	-1860.06	29←24,25	-17894.00	*6←4,5	-175946.62
*7←6,19,37	-1860.71	39←22	-17895.44	*34←10,36	-175984.94
*53←9,34,54	-1861.32	*10←5,27,58	-17899.47	*9←8,13	-176040.59
*44←20,22,42	-1861.37	*42←0,6,51	-17901.33	*27←0,22,24	-176061.64
*59←6,25	-1862.25	6←13	-17901.85	59←25	-176073.28
12←43	-1862.25	*38←0,6,24	-17902.48	*50←0,8	-176143.16
5←12	-1862.51	*53←3,34,56	-17907.16	31←24,25	-176176.78
*17←14,50,58	-1862.70	*8←5,9,14	-17908.33	*38←6,24,25	-176356.19
56←36	-1862.82	*2←0,49	-17908.99	Without	-176357.80
Without	-1893.34	Without	-17916.95	24←32	-177147.85

Appendix C

Table 16: The results of applying HCPK, ancestors relation, for synthetic data generated by the Insurance.

Insurance_100		**Insurance_1000**		**Insurance_10000**	
Optimal	-1686.22	Optimal	-13887.35	Optimal	-132968.57
25⇠ 4	-1691.67	11⇠22	-13973.78	0⇠25	-133689.72
23⇠6	-1691.67	8⇠3	-13978.52	10 ⇠ 9	-133710.57
18⇠13	-1693.71	2⇠3	-13979.52	24⇠5	-133714.13
16⇠13	-1694.28	12⇠3	-13986.60	13⇠2	-133793.24
7⇠6	-1695.01	14⇠1	-14005.63	2⇠25	-133834.40
18⇠15	-1695.07	21⇠3	-14011.25	26⇠15	-133852.04
15⇠13	-1695.34	0⇠4	-14011.47	11⇠6	-133870.28
19⇠17	-1695.42	16⇠15	-14022.36	22⇠1	-133879.29
22⇠4	-1695.44	17⇠4	-14023.35	19⇠7	-133902.66
24⇠15	-1695.49	14⇠6	-14025.36	7⇠ 1	-133950.60
20⇠1	-1695.49	10⇠4	-14029.36	23⇠22	-133972.61
19⇠7	-1695.50	Without	-14038.33	21⇠0	-133995.45
14⇠4	-1695.59	19⇠7	-14038.80	3⇠1	-134089.53
8⇠1	-1695.62	13⇠0	-14040.36	9⇠0	-134118.60
11⇠6	-1695.76	5⇠4	-14042.59	3⇠0	-134218.05
9⇢ 1	-1695.76	23⇠4	-14135.83	8⇠2	-134239.28
23⇠11	-1695.86	24⇠11	-14234.17	13⇠3	-134286.54
Without	-1697.81	25⇠0	-14294.73	Without	-134474.33
26⇠3	-1698.78	20⇠1	-14316.69	1 ⇠ 25	-134883.54
17⇠7	-1699.72	22⇠6	-14390.39	6 ⇠25	-136688.28

Table 17: The results of applying HCPK, ancestors relation, for synthetic data generated by the Alarm.

Alarm_100		**Alarm_1000**		**Alarm_10000**	
Optimal	-1349.22	Optimal	-11240.34	Optimal	-105226.51
25←···24	-1374.67	10←···7	-11403.12	4←···5	-105973.18
22←···24	-1377.11	21←···34	-11403.12	17←···16	-106052.34
36←···24	-1378.48	32←···18	-11410.48	10←···3	-106074.86
6←···7	-1379.44	28←···27	-11415.09	24←···23	-106093.55
22←···16	-1379.70	18←···10	-11428.23	26←···23	-106192.09
13←···10	-1380.30	2←···3	-11431.55	28←···27	-106202.46
33←···8	-1380.51	20←···3	-11433.07	18←···10	-106218.86
29←···8	-1381.10	17←···16	-11434.60	31←···3	-106276.17
32←···30	-1382.16	2←···7	-11435.07	7←···6	-106298.34
23←···24	-1383.34	35←···20	-11435.33	29←···2	-106316.37
34←···23	-1383.61	12←···3	-11443.31	36←···22	-106339.11
35←···19	-1383.66	15←···8	-11444.11	36←···22	-106339.11
4←···7	-1383.83	33←···2	-11457.36	27←···22	-106346.77
28←···17	-1383.88	5←···4	-11462.96	Without	-106496.20
Without	-1384.68	Without	-11467.32	33←···2	-106473.40
19←···0	-1384.97	9←···1	-11474.42	33←···2	-106473.40
27←···19	-1385.26	0←···11	-11484.00	21←···18	-106478.67
3←···9	-1385.57	25←···2	-11486.64	12←···8	-106509.12
11←···18	-1386.64	36←···25	-11496.21	32←···18	-106546.94
12←···0	-1387.08	22←···25	-11499.25	15←···8	-106669.29

Table 18: The results of applying HCPK, ancestors relation, for synthetic data generated by the Mildew.

Mildew _100		**Mildew _1000**		**Mildew _10000**	
Optimal	-6380.72	Optimal	-52258.85	Optimal	-454894.40
27⇠⋯26	-6579.86	20⇠⋯1	-52264.28	21⇠⋯20	-455394.63
27⇠⋯23	-6585.10	33⇠⋯31	-52285.45	24⇠⋯20	-455414.84
2⇠⋯21	-6592.21	22⇠⋯9	-52292.33	33⇠⋯14	-456072.11
7⇠⋯20	-6620.00	12⇠⋯31	-52303.90	30⇠⋯13	-456190.16
33⇠⋯32	-6643.30	9⇠⋯1	-52312.53	27⇠⋯9	-456259.62
Without	-6658.16	3⇠⋯27	-52399.22	30⇠⋯12	-456308.54
0⇠⋯11	-6658.16	4⇠⋯31	-52422.01	32⇠⋯31	-456589.06
3⇠⋯9	-6658.16	5⇠⋯7	-52423.05	3⇠⋯2	-456701.36
34⇠⋯33	-6658.16	8⇠⋯13	-52432.93	5⇠⋯8	-457011.69
32⇠⋯14	-6658.160	17⇠⋯15	-52443.92	16⇠⋯17	-457028.32
31⇠⋯14	-6658.16	21⇠⋯3	-52471.87	6⇠⋯8	-457053.18
32⇠⋯31	-6658.16	34⇠⋯14	-52475.47	20⇠⋯3	-457180.48
14⇠⋯9	-6658.16	Without	-52510.18	10⇠⋯11	-457479.49
4⇠⋯34	-6670.78	23⇠⋯3	-52524.10	14⇠⋯3	-458237.48
15⇠⋯0	-6670.78	19⇠⋯3	-52524.10	13⇠⋯4	-458284.12
15⇠⋯3	-6671.80	26⇠⋯9	-52313.50	26⇠⋯9	-458294.89
17⇠⋯19	-6672.05	9⇠⋯2	-52340.85	23⇠⋯8	-458727.85
29⇠⋯9	-6674.17	27⇠⋯9	-52292.33	27⇠⋯13	-459361.27
0⇠⋯12	-6691.10	14⇠⋯3	-52344.47	Without	-459642.31
4⇠⋯34	-6700.28	13⇠⋯14	-52297.75	21⇠⋯3	-460949.90

Table 19: The results of applying HCPK, ancestors relation, for synthetic data generated by the Carpo.

Carpo_100		**Carpo_1000**		**Carpo_10000**	
Optimal	-1829.30	Optimal	-17718.94	Optimal	-174130.56
54⇠4	-1853.74	53⇠0	-17807.83	1⇠24	-175438.70
11⇠17	-1857.04	27⇠6	-17813.72	27⇠38	-175449.54
9⇠0	-1857.07	48⇠14	-17824.20	50⇠24	-175484.91
41⇠13	-1858.26	5⇠16	-17824.88	38⇠4	-175684.80
14⇠35	-1858.38	14⇠16	-17828.84	52⇠6	-175758.50
0⇠35	-1859.22	41⇠34	-17831.81	37⇠32	-175886.53
20⇠13	-1859.51	18⇠5	-17840.14	13⇠16	-176006.31
26⇠4	-1860.42	16⇠20	-17840.57	45⇠34	-176055.59
21⇠34	-1860.50	19⇠12	-17842.66	21⇠11	-176060.13
55⇠35	-1860.68	0⇠13	-17844.01	18⇠19	-176074.25
53⇠3	-1860.74	30⇠13	-17845.99	17⇠11	-176108.39
49⇠33	-1860.84	10⇠12	-17849.95	44⇠26	-176158.73
3⇠24	-1861.45	44⇠33	-17852.52	57⇠11	-176178.40
59⇠0	-1862.63	24⇠12	-17853.11	16⇠6	-176209.82
30⇠2	-1862.97	13⇠14	-17854.83	58⇠0	-176275.47
5⇠43	-1864.16	59⇠24	-17870.56	29⇠20	-176355.47
29⇠1	-1864.40	37⇠32	-17888.63	Without	-176357.80
17⇠0	-1864.84	50⇠5	-17892.01	9⇠12	-176438.38
45⇠ 35	-1866.74	58⇠32	-17892.87	33⇠11	-176658.93
Without	-1893.34	Without	-17916.95	0⇠32	-176761.21

Appendix D

Conditional independence checks approach

Table 20: The results of applying HCPK, conditional independence, for synthetic data generated by the Insurance.

Insurance_100		**Insurance_1000**		**Insurance_10000**	
Optimal	-1686.22	Optimal	-13887.35	Optimal	-132968.57
{3,10} ⊥ 13 | 2	-1693.19	22 ⊥ 20| 1	-13970.8	10 ⊥ {4,6} | 9	-133635.10
5 ⊥ 7|26	-1693.73	3⊥ 7| ∅	-13992.87	25 ⊥ 5 | 6	-133640.68
11 ⊥ 14 | 6	-1694.62	1⊥ {2,12}| 4	-13997.25	12 ⊥ 15| 13	-133810.38
16 ⊥ 18,20|15	-1694.62	3⊥ 26| 15	-14000.51	11 ⊥ 21 | 1	-133834.78
13 ⊥ 21 |12	-1694.70	17 ⊥ 5 | 7	-14002.64	{9,8} ⊥ 14 | 1	-133861.81
12 ⊥ {19,23} | 13	-1695.49	8⊥ 6| 1	-14007.62	5 ⊥ 6 | ∅	-133866.53
6⊥ 9| 1	-1695.52	{5,1} ⊥ 24 | 16	-14010.52	21 ⊥ 22 | 0	-133891.00
5 ⊥ 10 | ∅	-1695.62	3⊥ 10| 4	-14016.64	{2,0} ⊥ 10 | 9	-133910.24
13 ⊥ 14 |15	-1695.76	3 ⊥ {22,11} | 4	-14019.42	20 ⊥ 5 | 12	-133968.49
1 ⊥ 11 | 6	-1695.86	20 ⊥ 24 | 16	-14030.33	25 ⊥ 6 | 1	-133969.31
10 ⊥ 12 | ∅	-1696.42	11⊥ 12| 4	-14034.82	24 ⊥ 26 | 15	-133974.57
7 ⊥ 22 | 4	-1696.52	Without	-14038.33	5 ⊥ 13 | ∅	-133977.21
2⊥ 12| ∅	-1696.76	15 ⊥ 24 | 16	-14040.12	0 ⊥ 15 | 13	-133983.46
3 ⊥ 21 | ∅	-1696.78	18⊥ 16| 15	-14042.80	4 ⊥ 20 | 1	-134057.24
15 ⊥ 20 | 1	-1696.78	{21,8} ⊥ 26 | 15	-14047.01	4 ⊥ 5 | ∅	-134188.78
16⊥ 18| 15	-1696.82	{9,2,}⊥ 6| 1	-14053.93	3 ⊥ 10 | 9	-134261.52
12 ⊥ 17 | 13	-1697.57	22 ⊥ 3 | ∅	-14055.88	1 ⊥ 12 | 0	-134291.56
Without	-1697.81	12 ⊥ 10 | ∅	-14066.43	{9,2} ⊥ 6 | 1	-134301.84
26 ⊥ 10 | 21	-1699.41	3 ⊥ 10 | ∅	-14074.55	Without	-134474.33
{8,10} ⊥ 6 | 1	-1701.61	14⊥ 16| 10	-14076.14	{21,12} ⊥ {22,7} | 0	-134702.23

Table 21: The results of applying HCPK, conditional independence, for synthetic data generated by the Alarm.

Alarm_100		**Alarm_1000**		**Alarm_10000**	
Optimal	-1349.22	Optimal	-11240.34	Optimal	-105226.51
31 ⊥ 1 | 0	-1373.52	0 ⊥ 15 | 10	-11384.56	19 ⊥ 26 | ∅	-105857.61
1 ⊥ 3 | 0	-1375.14	3 ⊥ {12,14} | 10	-11411.59	16 ⊥ 21 | 19	-105941.51
{16,14} ⊥ 3 | 9	-1377.07	0 ⊥ 7 | 2	-11420.83	2 ⊥ 3 | ∅	-106106.84
{26,6} ⊥ 34 | 23	-1379.02	14 ⊥ 2 | 33	-11421.83	6 ⊥ 29 | 7	-106126.18
35 ⊥ 24 | 25	-1380.15	1 ⊥ 30 | 0	-11431.38	31 ⊥ 9 | 3	-106130.97
32 ⊥ 27 | 19	-1380.80	3 ⊥ 12 | 10	-11432.22	34 ⊥ {24,26} | 23	-106208.56
26 ⊥ 16 | ⌀	-1381.83	10 ⊥ 11 | ∅	-11432.35	17 ⊥ 15 | ∅	-106267.41
24 ⊥ {35,26 }|22	-1382.01	8 ⊥ {25,14} | 2	-11436.23	{6,14} ⊥ {29,15} | 7	-106313.33
25 ⊥ 36 | 24	-1382.07	6 ⊥ 31 | 7	-11444.39	0 ⊥ 3 | ∅	-106352.21
9 ⊥ 7 | ∅	-1382.21	25 ⊥ 23 | 24	-11447.10	19 ⊥ 20 | ∅	-106427.13
17 ⊥ 9 | 16	-1382.95	10 ⊥ 22 | 2	-11450.03	16 ⊥ {28,27} | 17	-106474.39
24 ⊥ 34 | 23	-1383.20	15 ⊥ 12 | 10	-11450.35	Without	-106496.20
28 ⊥ 25 | 22	-1383.31	{23,34} ⊥ 36 | 24	-11452.67	9 ⊥ 1 | 0	-106529.75
8 ⊥ 31|7	-1383.83	{6,3} ⊥ {1,25} | 2	-11453.78	{0,1} ⊥ {8,6} | 3	-106549.80
14 ⊥ 17 | ∅	-1383.90	30 ⊥ 9 | 0	-11463.15	0 ⊥ 8 | 19	-106563.69
0 ⊥ 19 | 18	-1383.97	{17,16} ⊥ 3 | 10	-11463.65	29 ⊥ 0 |12	-106572.44
Without	-1384.68	Without	-11467.32	7 ⊥ 4 | 5	-106635.78
16 ⊥ 7 | 1	-1385.19	{2,3} ⊥ 11 | ∅	-11467.57	{8,7} ⊥ 9 | 3	-106637.25
{18,19} ⊥ 25 | 24	-1385.34	31 ⊥ 24 | 25	-11470.19	19 ⊥ 29 | 18	-106653.91
1 ⊥ 17 | ∅	-1385.86	3 ⊥ 7 | ∅	-11471.48	5 ⊥ 6 | ∅	-107342.07

Table 22: The results of applying HCPK, conditional independence, for synthetic data generated by the Mildew.

Mildew _100		**Mildew _1000**		**Mildew _10000**	
Optimal	-6380.72	Optimal	-52258.85	Optimal	-454894.40
Without	-6658.16	0 ⊥ 10 | 11	-52292.33	{1,3} ⊥ {25,26} | 14	-455587.36
23 ⊥ {26, 24} | φ	-6658.16	6 ⊥ 5 | 7	-52292.33	28 ⊥ 31 | φ	-455653.01
27 ⊥ 7 | 23	-6658.16	30 ⊥ 4 | 13	-52292.33	1 ⊥ 25 | 9	-455849.04
13 ⊥ 20 | 12	-6658.16	31 ⊥ 26 | 14	-52292.33	16 ⊥ 15 | 17	-455868.95
12 ⊥ {27, 29} | 11	-6658.16	12 ⊥ 4 | 13	-52292.33	8 ⊥ 12 | φ	-456028.60
30 ⊥ 34 | φ	-6658.16	13 ⊥ 29 | φ	-52297.75	14 ⊥ 25 | φ	-456201.06
14 ⊥ 21 | 9	-6658.16	{2,3} ⊥ {26,27} | 14	-52297.75	7 ⊥ 13 | 8	-456312.73
22 ⊥ 24 | φ	-6658.16	1 ⊥ 20 | 3	-52297.75	3 ⊥ 26 | 14	-456362.77
19 ⊥ 15 | 17	-6658.16	21 ⊥ 19 | 20	-52297.75	13 ⊥ 34 | 14	-456379.62
14 ⊥ 32 | 31	-6658.16	{2,9} ⊥ 21 | 20	-52304.02	6 ⊥ 5 | 7	-456503.22
23 ⊥ 5 | 7	-6658.16	3 ⊥ 23 | 9	-52304.02	0 ⊥ {16,18} | φ	-456564.09
34 ⊥ 8 | 27	-6658.16	16 ⊥ 15 | 17	-52307.60	9 ⊥ 20 | 3	-456611.20
{1,6} ⊥ {16,18} | φ	-6662.71	{16,18} ⊥ {15,25} | 17	-52307.60	0 ⊥ 10 | 11	-456642.36
23 ⊥ 24 | φ	-6665.10	2 ⊥ 26 | 14	-52312.21	6 ⊥ {5,31} | 7	-457044.02
22 ⊥ 3 | 9	-6665.10	18 ⊥ 24 | φ	-52312.21	{12,18} ⊥ {33,34} | 14	-457147.47
28 ⊥ 0 | 11	-6665.10	3 ⊥ 34 | 9	-52313.50	{19,18} ⊥ 8 | 3	-457160.45
9 ⊥ 31 | 14	-6665.10	{3,9} ⊥ 22 | 23	-52322.06	2 ⊥ 5 | 8	-457542.56
9 ⊥ 32 | 31	-6665.10	1 ⊥ 2 | φ	-52325.98	29 ⊥ 22 | 30	-457799.43
1 ⊥ 6 | φ	-6665.10	14 ⊥ 4 | 31	-52357.31	4 ⊥ 2 | φ	-458315.63
13 ⊥ 20 | 31	-6665.10	Without	-52510.18	Without	-459642.31

Table 23: The results of applying HCPK, conditional independence, for synthetic data generated by the Carpo.

Carpo_100		**Carpo_1000**		**Carpo_10000**	
Optimal	-1829.30	Optimal	-17718.94	Optimal	-174130.56
$18 \perp 35 \mid 12$	-1857.66	$46 \perp 40 \mid \phi$	-17803.26	$\{30,3\} \perp \{2,58\} \mid 0$	-175154.78
$43 \perp 13 \mid 12$	-1857.72	$12 \perp 15 \mid \phi$	-17811.24	$\{48, 34\} \perp 4 \mid 35$	-175211.78
$14 \perp 58 \mid 0$	-1859.97	$52 \perp 38 \mid 6$	-17819.98	$24 \perp 8 \mid \phi$	-175349.54
$5 \perp 43 \mid 12$	-1860.41	$1 \perp \{44,38\} \mid 0$	-17824.69	$34 \perp 4 \mid 35$	-175351.80
$34 \perp 21 \mid 20$	-1860.65	$\{39,55\} \perp 54 \mid 22$	-17825.13	$2 \perp 3 \mid 0$	-175411.19
$21 \perp 1 \mid 20$	-1860.78	$56 \perp 28 \mid 20$	-17826.58	$6 \perp 24 \mid \phi$	-175526.15
$13 \perp 35 \mid \phi$	-1860.92	$48 \perp 23 \mid 34$	-17830.17	$11 \perp 39 \mid 20$	-175542.47
$35 \perp 18 \mid \phi$	-1861.31	$14 \perp 40 \mid 16$	-17830.28	$10 \perp 36 \mid \phi$	-175572.90
$18 \perp 34 \mid 13$	-1861.84	$55 \perp 54 \mid 22$	-17830.73	$35 \perp 4 \mid 45$	-175673.73
$3 \perp 33 \mid 0$	-1862.04	$7 \perp 48 \mid 34$	-17832.27	$30 \perp 2 \mid 0$	-175687.69
$13 \perp 18 \mid \phi$	-1862.17	$1 \perp 44 \mid 0$	-17841.24	$\{29, 32\} \perp 37 \mid 24$	-175691.73
$\{18,35\} \perp 36 \mid 34$	-1862.40	$32 \perp 37 \mid 24$	-17841.39	$\{1,46\} \perp 58 \mid 0$	-175702.74
$\{58,3\} \perp 2 \mid 0$	-1862.76	$47 \perp 37 \mid 24$	-17842.08	$1 \perp 58 \mid 0$	-175716.66
$24 \perp 3 \mid 0$	-1862.89	$\{11,58\} \perp 57 \mid 20$	-17843.05	$18 \perp 29 \mid 19$	-175924.69
$10 \perp 36 \mid \phi$	-1863.01	$57 \perp 33 \mid 20$	-17844.96	$37 \perp 33 \mid 24$	-175948.52
$36 \perp 57 \mid 56$	-1863.19	$10 \perp 18 \mid 19$	-17858.55	$20 \perp 16 \mid 11$	-175983.38
$18 \perp 36 \mid 34$	-1863.59	$11 \perp 12 \mid \phi$	-17861.97	$6 \perp 12 \mid 14$	-176082.25
$\{13,35\} \perp 18 \mid \phi$	-1869.13	$1 \perp 38 \mid 0$	-17882.78	$\{14,11\} \perp 15 \mid \phi$	-176210.61
$20 \perp 23 \mid 33$	-1872.09	$15 \perp 11 \mid \phi$	-17887.52	$11 \perp 14 \mid \phi$	-176224.26
Without	-1893.34	Without	-17916.95	Without	-176357.80

Backtrack approach

Table 24: The results of applying HCPK, conditional independence, for synthetic data generated by the Insurance.

Insurance_100		**Insurance_1000**		**Insurance_10000**	
Optimal	-1686.22	Optimal	-13887.35	Optimal	-132968.57
11 ⊥ 14 | 6	-1691.65	3⊥ 26| 15	-13961.60	1 ⊥ 12 | 0	-133706.43
3 ⊥ 21 | ∅	-1694.62	11⊥ 12| 4	-13967.58	0 ⊥ 15 | 13	-133717.48
13 ⊥ 21 |12	-1694.72	3⊥ 7| ∅	-13978.71	20 ⊥ 5 | 12	-133800.31
16 ⊥ 18,20|15	-1695.32	18⊥ 16| 15	-13999.65	5 ⊥ 13 | ∅	-133812.14
12 ⊥ {19,23} | 13	-1695.34	22 ⊥ 3 | ∅	-14001.24	10 ⊥ {4,6} | 9	-133827.87
10 ⊥ 12 | ∅	-1695.44	{21,8} ⊥ 26 | 15	-14011.20	3 ⊥ 10 | 9	-133849.94
2⊥ 12| ∅	-1695.49	20 ⊥ 24 | 16	-14011.49	4 ⊥ 20 | 1	-133855.60
6⊥ 9| 1	-1695.50	{5,1} ⊥ 24 | 16	-14013.06	21 ⊥ 22 | 0	-133864.14
15 ⊥ 20 | 1	-1695.50	14⊥ 16| 10	-14014.60	{9,8} ⊥ 14 | 1	-133907.63
16⊥ 18| 15	-1695.52	15 ⊥ 24 | 16	-14020.57	25 ⊥ 6 | 1	-133935.88
5 ⊥ 7| 26	-1695.59	22 ⊥ 20| 1	-14022.39	25 ⊥ 5 | 6	-133935.88
5 ⊥ 10 | ∅	-1695.59	3 ⊥ 10 | ∅	-14023.12	12 ⊥ 15| 13	-133975.39
26 ⊥ 10 | 21	-1695.76	3⊥ 10| 4	-14023.16	11 ⊥ 21 | 1	-134011.44
{3,10} ⊥ 13 | 2	-1695.76	8⊥ 6| 1	-14025.73	4 ⊥ 5 | ∅	-134063.19
1 ⊥ 11 | 6	-1695.86	1⊥ {2,12}| 4	-14026.38	{21,12} ⊥ {22,7} | 0	-134105.09
12 ⊥ 17 | 13	-1697.26	17 ⊥ 5 | 7	-14029.44	24 ⊥ 26 | 15	-134127.88
13 ⊥ 14 |15	-1697.49	Without	-14038.33	5 ⊥ 6 | ∅	-134175.76
Without	-1697.81	3 ⊥ {22,11} | 4	-14042.36	{2,0} ⊥ 10 | 9	-134324.92
{8,10} ⊥ 6 | 1	-1702.40	{9,2,}⊥ 6| 1	-14054.60	{9,2} ⊥ 6 | 1	-134437.74
7 ⊥ 22 | 4	-1706.62	12 ⊥ 10 | ∅	-14059.09	Without	-134474.33

Table 25: The results of applying HCPK, conditional independence, for synthetic data generated by the Alarm.

Alarm_100		**Alarm_1000**		**Alarm_10000**	
Optimal	-1349.22	Optimal	-11240.34	Optimal	-105226.51
25 ⊥ 36 | 24	-1374.55	3 ⊥ {12,14} | 10	-11380.49	34 ⊥ {24,26} | 23	-105893.03
8 ⊥ 31|7	-1376.04	{6,3} ⊥ {1,25} | 2	-11401.14	5 ⊥ 6 | ∅	-105907.83
14 ⊥ 17 | ∅	-1378.69	25 ⊥ 23 | 24	-11405.91	0 ⊥ 8 | 19	-105948.84
{16,14} ⊥ 3 | 9	-1380.23	{17,16} ⊥ 3 | 10	-11412.46	11 ⊥ 10 | ∅	-106030.56
{26,6} ⊥ 34 | 23	-1380.43	10 ⊥ 11 | ∅	-11412.60	19 ⊥ 26 | ∅	-106102.35
24 ⊥ {35,26 }|22	-1380.62	30 ⊥ 9 | 0	-11420.60	19 ⊥ 29 | 18	-106175.85
16 ⊥ 7 | 1	-1380.78	6 ⊥ 31 | 7	-11421.98	7 ⊥ 4 | 5	-106177.47
9 ⊥ 7 | ∅	-1381.84	0 ⊥ 7 | 2	-11423.43	16 ⊥ {28,27} | 17	-106230.54
14 ⊥ 5 | 16	-1381.84	10 ⊥ 22 | 2	-11431.22	2 ⊥ 3 | ∅	-106234.82
31 ⊥ 1 | 0	-1383.04	14 ⊥ 2 | 33	-11432.35	16 ⊥ 21 | 19	-106235.67
35 ⊥ 24 | 25	-1383.16	3 ⊥ 12 | 10	-11435.66	9 ⊥ 1 | 0	-106265.50
26 ⊥ 16 | ⌀	-1383.37	15 ⊥ 12 | 10	-11439.40	6 ⊥ 29 | 7	-106278.46
17 ⊥ 9 | 16	-1383.67	31 ⊥ 24 | 25	-11442.60	0 ⊥ 3 | ∅	-106285.84
Without	-1384.68	0 ⊥ 15 | 10	-11445.44	{6,14} ⊥ {29,15} | 7	-106289.23
1 ⊥ 3 | 0	-1385.34	5 ⊥ 6 | ∅	-11447.34	29 ⊥ 0 |12	-106289.66
32 ⊥ 27 | 19	-1385.36	8 ⊥ {25,14} | 2	-11454.67	{0,1} ⊥ {8,6} | 3	-106295.53
24 ⊥ 34 | 23	-1385.87	1 ⊥ 30 | 0	-11456.61	17 ⊥ 15 | ∅	-106314.16
{18,19} ⊥ 25 | 24	-1385.95	{2,3} ⊥ 11 | ∅	-11462.93	Without	-106496.20
28 ⊥ 25 | 22	-1385.98	Without	-11467.32	31 ⊥ 9 | 3	-106590.23
0 ⊥ 19 | 18	-1388.70	{23,34} ⊥ 36 | 24	-11474.42	{8,7} ⊥ 9 | 3	-106602.64
1 ⊥ 17 | ∅	-1389.75	3 ⊥ 7 | ∅	-11476.70	19 ⊥ 20 | ∅	-106685.48

Table 26: The results of applying HCPK, conditional independence, for synthetic data generated by the Mildew.

Mildew _100		**Mildew _1000**		**Mildew _10000**	
Optimal	-6380.72	Optimal	-52258.85	Optimal	-454894.40
Without	-6658.16	0 ⊥ 10 | 11	-52292.33	{1,3} ⊥ {25,26} | 14	-455587.36
23 ⊥ {26, 24} | φ	-6658.16	6 ⊥ 5 | 7	-52292.33	28 ⊥ 31 | φ	-455653.01
27 ⊥ 7 | 23	-6658.16	30 ⊥ 4 | 13	-52292.33	1 ⊥ 25 | 9	-455849.04
13 ⊥ 20 | 12	-6658.16	31 ⊥ 26 | 14	-52292.33	16 ⊥ 15 | 17	-455868.95
12 ⊥ {27, 29} | 11	-6658.16	{2,9} ⊥ 21 | 20	-52292.33	8 ⊥ 12 | φ	-456028.60
30 ⊥ 34 | φ	-6658.16	12 ⊥ 4 | 13	-52292.33	14 ⊥ 25 | φ	-456201.06
14 ⊥ 21 | 9	-6658.16	13 ⊥ 29 | φ	-52297.75	7 ⊥ 13 | 8	-456312.73
22 ⊥ 24 | φ	-6658.16	{2,3} ⊥ {26,27} | 14	-52297.75	3 ⊥ 26 | 14	-456362.77
19 ⊥ 15 | 17	-6658.16	1 ⊥ 20 | 3	-52297.75	13 ⊥ 34 | 14	-456379.62
14 ⊥ 32 | 31	-6658.16	21 ⊥ 19 | 20	-52297.75	6 ⊥ 5 | 7	-456503.22
23 ⊥ 5 | 7	-6658.16	3 ⊥ 23 | 9	-52304.02	0 ⊥ {16,18} | φ	-456564.09
34 ⊥ 8 | 27	-6658.16	16 ⊥ 15 | 17	-52307.60	9 ⊥ 20 | 3	-456611.20
{1,6} ⊥ {16,18} | φ	-6662.71	{16,18} ⊥ {15,25} | 17	-52307.60	0 ⊥ 10 | 11	-456642.36
23 ⊥ 24 | φ	-6665.10	2 ⊥ 26 | 14	-52312.21	6 ⊥ {5,31} | 7	-457044.02
22 ⊥ 3 | 9	-6665.10	3 ⊥ 34 | 9	-52313.50	{12,18} ⊥ {33,34} | 14	-457147.47
28 ⊥ 0 | 11	-6665.10	{3,9} ⊥ 22 | 23	-52322.06	{19,18} ⊥ 8 | 3	-457160.45
9 ⊥ 31 | 14	-6665.10	1 ⊥ 2 | φ	-52325.98	2 ⊥ 5 | 8	-457542.56
9 ⊥ 32 | 31	-6665.10	14 ⊥ 4 | 31	-52357.31	29 ⊥ 22 | 30	-457799.43
1 ⊥ 6 | φ	-6665.10	14 ⊥ {4,13} | 31	-52360.22	4 ⊥ 2 | φ	-458315.63
13 ⊥ 20 | 31	-6665.10	Without	-52510.18	Without	-459642.31

Appendix E

Table 27: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Insurance.

Insurance_100		**Insurance_1000**		**Insurance_10000**	
Optimal	-1686.22	Optimal	-13887.35	Optimal	-132968.57
Without	-1697.81	Without	-14038.33	8 ⊥ 10| 7	-134175.69
12⇠13	-1697.81	20 ⊥ 13 | ∅	-14269.85	Without	-134474.33
13 ⊥ {21,24} | 26	-1700.52	4⇠22	-14279.55	5⇠7	-137956.77
18⇠19	-1701.94	10⇠17	-14332.42	26⇢16	-138946.45
22 ⇠23	-1702.79	11←16	-14365.53	0⇠3	-140432.56
26←5	-1702.83	3←0,2	-14368.29	26←21,7	-141171.52
10 ⇠13	-1702.95	2←1,8	-14368.79	{15,8} ⊥ 18| 19	-141318.94
0 ⊥ 2 | 26	-1703.01	14 ⊥ 15 | 3	-14387.19	5⇠1,3	-142031.43
10 ⊥ {21,26} | 0	-1704.97	22 ⊥ 26 | 16	-14435.432	10←0	-142134.12
4 ⊥ 13 | ∅	-1721.12	0⇠25	-14487.30	1⇠4	-142176.65
13 ← 22,9	-1721.42	25 ⊥ {12,11} | 6	-14498.98	0 ⊥ 15| 12	-142571.83
7 ← 22,13	-1737.38	11⇠23	-14611.14	5←15,13	-143289.97
0⇐26,9,4	-1746.95	17←3,7,1	-14733.58	{17,6} ⊥ 13 | 2	-144445.43

Table 28: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Alarm.

Alarm_100		**Alarm_1000**		**Alarm_10000**	
Optimal	-1349.22	Optimal	-11240.34	Optimal	-105226.51
10⇠12	-1384.23	11←2,1	-11458.50	Without	-106376.13
Without	-1384.68	Without	-11467.32	{7,11}⊥3| 35	-113744.13
10⊥{21,20}|∅	-1386.69	3 ← 6	-11467.61	26←19,27	-114776.71
2←14	-1389.06	20←19,21	-11475.40	20⇠35	-115153.53
24⇠36	-1389.11	3⇠30	-11490.28	30⇠32	-115642.70
3⊥13| 9	-1389.89	14⇠30	-11514.29	2⇠31	-115939.97
7 ⊥ 4 | 5	-1390.74	31⇠2	-11525.48	15⊥9| 27	-117195.50
14⇠28	-1391.50	11⇠31	-11526.02	23 ⊥ {25,17} | 22	-117439.74
7⇠30	-1394.72	27←17,28	-11670.61	9←13,12	-117853.31
6←33,9	-1395.36	33⊥1| 16	-12065.96	24←36	-117909.42
35←0,7,33	-1405.77	30⊥6| ∅	-12221.95	11⇠12	-118255.05
15←5,12,22	-1411.10	22⊥20| 16	-12371.76	3←0,9	-118387.56
{13,20} ⊥ 30 | 7	-1468.11	2⊥{12,4}| 16	-12500.21	20⊥21| 27	-118456.34

Table 29: The results of applying HCPK, inconsistent prior knowledge, for synthetic data generated by the Mildew.

Mildew _100		**Mildew _1000**		**Mildew _10000**	
Optimal	-6380.72	Optimal	-52258.85	Optimal	-454894.40
34 ⇠4	-6605.61	6 ⊥ 7 | 5	-52333.38	1⇠3	-455959.98
27 ← 8,9,34	-6623.89	30 ⊥ 27 | ∅	-52337.50	19⇠20	-456768.16
Without	-6658.16	3←5	-52348.57	3← 4,5	-456879.13
27 ← 7	-6658.16	22 ⊥ 20 | 16	-52367.27	20 ⊥ 6 | 21	-456937.74
20 ⊥ 26 | 2	-6658.16	{4,8} ⊥ 31 | 13	-52423.58	33 ⊥ 12 | ∅	-457328.70
23 ⊥ {7, 27} | 12	-6658.16	24⇠5	-52495.74	29 ⊥ 4 | 30	-457905.55
{ 34,26} ⊥ {19,5} | 3	-6658.16	Without	-52510.18	3⇠14	-457942.74
27 ⊥ 11 | 29	-6658.16	2⇠12	-52624.50	4← 8,9	-457951.11
32 ⇠33	-6662.86	14⇠26	-52634.89	{1,3} ⊥ 27 | 9	-457973.62
19 ⇠17	-6672.05	22←30,2	-52654.26	8⇠13	-458002.35
23⇠4	-6677.38	7⇠12	-52716.42	4⇠8	-458107.37
2 ← 0,2,34	-6750.20	30←2,3	-52846.96	30← 32,1	-459603.47
33 ← 12, 13	-6876.71	8←20,1	-53779.44	Without	-459642.31

Appendix F

Table 30: The results of applying HCPK for synthetic data generated by the Diabetes.

Diabetes					
Arrows which have to be there	BNs score	Ancestor relation	BNs	Conditional independence checks	BNs score
59←57,58	58192.49	11⇠8	-58206.53	$391 \perp 388 \mid 389$	-58198.89
358←357,54	-58046.47	312⇠304	-58206.53	$\{9,2\} \perp 6 \mid 1$	-58206.53
199←45,198	-58054.52	401⇠394	-58211.55	$90 \perp 9 \mid 10$	-58206.53
373←368,372	-58083.91	344⇠343	-58218.09	$2 \perp 1 \mid \emptyset$	-58206.53
397←57,396	-58109.54	230⇠228	-58231.35	$228 \perp 254 \mid \emptyset$	-58206.53
409←392,408	-58109.84	12⇠10	-58234.26	$404 \perp 412 \mid 410$	-58206.53
81←68,80	-58156.93	209⇠2758	-58265.46	$379 \perp 402 \mid 380$	-58206.53
401←395,400	-58186.88	91⇠14	-58476.69	$307 \perp 310 \mid 164$	-58206.53
148←147,144	-58192.98	393⇠391	-58635.04	$55 \perp 241 \mid 30$	-58206.53
302←287,288	-58205.88	410⇠402	-58733.17	$204 \perp 202 \mid 205$	-58206.53
120←107,119	-58206.42	165⇠162	-58877.96	$176 \perp 177 \mid 175$	-58215.17
410←404,405	-58211.44	52⇠0	-58877.96	$43 \perp 46 \mid 44$	-58218.34
277←275	-58214.05	377⇠359	-58877.96	$175 \perp 195 \mid 188$	-58225.77
319←317,318	-58230.24	187⇠42	-59483.24	$324 \perp 310 \mid \emptyset$	-58292.40
171←170,169	-58230.28	355⇠331	-59506.82	$\{263,247\} \perp 279 \mid 57$	-58326.58
135←130	-58242.94	20-6	-59508.72	$306 \perp 307 \mid \emptyset$	-58340.79
70←68	-58248.32	261⇠259	-59543.73	$58 \perp 57 \mid \emptyset$	-58357.53
300←299,298	-58292.40	101⇠99	-59683.41	$216 \perp 232 \mid 57$	-58764.96
282←275,280	-58303.33	41⇠40	-59888.84	$40 \perp 39 \mid \emptyset$	-58877.96
250←243,248	-58318.47	180⇠57	-60150.23	$57 \perp 167 \mid \emptyset$	-58877.96
Without	-59695.4	Without	-62506.72	Without	-59695.4

Bibliography

Angelopoulos, N., & Cussens, J. (2005). Exploiting Informative Priors for Bayesian Classification and Regression Trees. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 641-646, Edinburgh

Angelopoulos, N., & Cussens, J. (2009). Bayesian learning of Bayesian networks with informative priors. *Annals of Mathematics and Artificial Intelligence*, *54*(1-3), 53–98. doi:10.1007/s10472-009-9133-x

Atallah, M., & Blanton, M. (Eds.). (2009). *Algorithms and Theory of Computation Handbook* (Second Edi., Vol. 4). Chapman and Hall/CRC. doi:10.1201/9781584888215

Berger, J. O., Pericchi, L. R., Ghosh, J., Samanta, T., De Santis, F., Berger, J., & Pericchi, L. (2001). Objective Bayesian methods for model selection: introduction and comparison. *Lecture Notes-Monograph Series*, *38*(2001), 135–207.

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)* (p. 740). Springer.

Borboudakis, G., & Tsamardinos, I. (2012). Incorporating Causal Prior Knowledge as Path-Constraints in Bayesian Networks and Maximal Ancestral Graphs. *arXiv Preprint arXiv:1206.6390*.

Buntine, W. (1991). Theory refinement on Bayesian networks. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence* (Vol. 91).

Campos, C. De, Zeng, Z., & Ji, Q. (2009). Structure learning of Bayesian networks using constraints. *Conference on Machine Learning*.

Castelo, R. V., & Siebes, A. (1998). Priors on network structures. Biasing the search for Bayesian networks. , International Journal of Approximate Reasoning, 24 (1), pp. 39-57

Chickering, D., Geiger, D., & Heckerman, D. (1995). Learning Bayesian networks: Search methods and experimental results. In *Proceedings of Fifth Conference on Artificial Intelligence and Statistics* (pp. 112–128).

Chickering, D., Heckerman, D., & Meek, C. (2003). Large-sample learning of Bayesian networks is NP-hard. *Of Machine Learning*, *5*, 1287–1330.

Chickering, D. M. (2002). Learning equivalence classes of Bayesian-network structures. *The Journal of Machine Learning Research*, *2*, 445–498.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, *42*(2-3), 393–405. doi:10.1016/0004-3702(90)90060-D

Cooper, G. F., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*(4), 309–347. doi:10.1007/BF00994110

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (p. 1312). MIT Press; 3rd Revised edition edition.

Cowell, R. G. (2009). Efficient maximum likelihood pedigree reconstruction. *Theoretical Population Biology*, *76*(4), 285–291. doi:10.1016/j.tpb.2009.09.002

Cussens, J. (2011). Bayesian network learning with cutting planes.

Friedman, N. (1996). Building classifiers using Bayesian networks. *Proceedings of the National Conference on*, 1277–1284.

Friedman, N. (1999). Learning Bayesian Network Structure from Massive Datasets : The " Sparse Candidate " Algorithm. *Science*, 206–215.

Friedman, N., & Koller, D. (2003). Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, *50*(1), 95–125.

Friedman, N., Murphy, K., & Russell, S. (1998). Learning the Structure of Dynamic Probabilistic Networks. *Science*, 139–147.

Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (2003). *Bayesian Data Analysis (Chapman & Hall/CRC Texts in Statistical Science)* (p. 696). Chapman and Hall/CRC.

Getoor, L., & Taskar, B. (2007). *Introduction to Statistical Relational Learning* (p. 796). MIT Press.

Gillispie, S. B., & Perlman, M. D. (2001). Enumerating Markov equivalence classes of acyclic digraph models. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence* (pp. 171–177). Morgan Kaufmann Publishers Inc.

Heckerman, D. (1996). A Tutorial on Learning With Bayesian Networks, *1995*(November).

Heckerman, D., & Chickering, D. M. (1995). Learning Bayesian Networks : The Combination of Knowledge and Statistical Data Metrics for Belief Networks. *Machine Learning 20*.

Jensen, F. V, & Nielsen, T. D. (2007). *Bayesian networks and decision graphs. Statistics for engineering and information science* (second.). Springer Verlag.

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*.

Koller, D., Friedman, N., Getoor, L., & Taskar, B. (2007). Graphical Models in a Nutshell. *Graphical Models*.

Korb, K. B., & Ann E. Nicholson. (2003). Introducing Bayesian Networks. *Bayesian Artificial Intelligence*, 29–54.

Krause, P. J. (1998). Learning probabilistic networks. *Engineering*, *13*(1994).

Mansinghka, V. K., Kemp, C., Tenenbaum, J. B., & Griffiths, T. L. (2006). Structured Priors for Structure Learning. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (UAI)*.

Markowetz, F., & Spang, R. (2007). Inferring cellular networks--a review. *BMC Bioinformatics*, *8 Suppl 6*, S5. doi:10.1186/1471-2105-8-S6-S5

Murphy, K. P. (2001). An introduction to graphical models, (May), 1–19.

Nilsson, N. J. (2005). INTRODUCTION TO MACHINE LEARNING AN EARLY DRAFT OF A PROPOSED TEXTBOOK Department of Computer Science. *Machine Learning*.

Richard E. Neapolitan. (2004). *Learning Bayesian Networks*.

Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (third edit.). prentice hell.

Sheehan, N. A., & Egeland, T. (2007). Structured Incorporation of Prior Information in Relationship Identification Problems. *Annals of Human Genetics*, *71*(4), 501–518.

Silander, T., Kontkanen, P., & Myllymäki, P. (2007). On sensitivity of the MAP Bayesian network structure to the equivalent sample size parameter. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence* (pp. 360–367). Citeseer.

Silander, T., & Myllymäki, P. (2006). A simple approach for finding the globally optimal Bayesian network structure. *Networks*.

Tsamardinos, I., Brown, L. E., & Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, *65*(1), 31–78. doi:10.1007/s10994-006-6889-7

Werhli, A., & Husmeier, D. (2007). Reconstructing gene regulatory networks with Bayesian networks by combining expression data with multiple sources of prior knowledge. *Statistical Applications in Genetics and Molecular Biology, 6(1), Art. 15.*