1-1-2010

A Heuristic Search Algorithm for Learning Optimal Bayesian Networks

Xiaojian Wu

A HEURISTIC SEARCH ALGORITHM FOR LEARNING OPTIMAL BAYESIAN NETWORKS

By

Xiaojian Wu

A Thesis Proposal
Submitted to the Faculty of
Mississippi State University
in Partial Fulf llment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

August 2010

A HEURISTIC SEARCH ALGORITHM FOR LEARNING

OPTIMAL BAYESIAN NETWORKS

By

Xiaojian Wu

Approved:

Changhe Yuan Assistant Professor of Computer Science and Engineering (Major Professor)	Eric A. Hansen Associate Professor of Computer Science and Engineering (Committee Member)
Russell Stocker Assistant Professor of Mathematics and Statistics (Committee Member)	Edward B. Allen Associate Professor of Computer Science and Engineering, and Graduate Coordinator

Sarah A. Rajala
Dean of the James Worth Bagley College of Engineering

Name: Xiaojian Wu

Date of Degree: August 07, 2010

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Changhe Yuan

Director of Thesis Proposal: Dr. Changhe Yuan

Title of Study: A HEURISTIC SEARCH ALGORITHM FOR LEARNING OPTIMAL BAYESIAN NETWORKS

Pages in Study: 45

Candidate for Degree of Master of Science

Bayesian network is a popular machine learning tool for modeling uncertain dependence relationships among the random factors of a domain. It represents the relations qualitatively by using a directed acyclic graph (DAG) and quantitatively by using a set of conditional probability distributions. Several exact algorithms for learning optimal Bayesian networks from data have been developed recently. However, these algorithms are still ineff cient to some extent. This is not surprising because learning Bayesian network has been proven to be an NP-Hard problem. Based on a critique of these algorithms, this thesis introduces a new algorithm based on heuristic search for learning optimal Bayesian networks. Empirical results show that this new algorithm is more eff cient than the existing algorithms.

Key words: Bayesian Network, Directed Acyclic Graph, Structure Learning, Heursitic Search, Dynamic Programming

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation grant IIS-0953723. I thank Dr. Changhe Yuan for directing me in this research. I also thank Drs. Eric A. Hansen and Russell Stocker for serving on my thesis committee.

Some f gures in this thesis were generated using GeNIe, a software developed by the University of Pittsburgh.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In statistics, the probability of the occurrence of a random event has two different interpretations. One interpretation is that probability is a physical property of the event. For example, a probability 0.5 with which a coin lands head means that, when the total number of tosses goes to inf nity, the proportion of heads will converge to 0.5. Another interpretation is that probability is a person's degree of belief on events [16]. In this interpretation, a probability 0.5 represents a person's degree of belief on the coin landing head in the next toss, i.e., probability is not a physical property but just a person's belief. This belief may originate from the person's daily experience or from other people's suggestions. In this thesis, I focus on the latter interpretation of probability.

So far, we only consider the probability of a single random event. There are many cases in which we are interested in modeling a more complex system that contains more than one relevant random factors. To model multiple correlated random factors, a general solution is to use a joint probability distribution. But this representation is impractical for large domains as it requires too much memory for storing the joint distributions that have large dimensions. This representation does not model conditional independence relations that may be present among the variables. In the past several decades, Bayesian networks

have been used to provide a compact representation of joint probability distributions by explicitly modeling independence relationships.

Many approaches have been developed to construct Bayesian networks. One easy approach is to translate experts' knowledge of a domain into casual relationships among the random factors, which are then used to build the Bayesian networks. Currently, this method has been used in medicine, economy, psychology, etc. However, working with experts is somewhat ineff cient. Plus, there is often insuff cient domain knowledge available to build a complete Bayesian network. This is typically true, for instance, for newly emerged research areas such as biology and chemistry. Therefore, it is necessary to develop effective machine learning methods for learning Bayesian networks from data. In the remainder of this chapter, I will give a more detailed discussion on Bayesian network and its learning methods.

1.1 Overview of Bayesian Network

For a domain with n variables $\mathbf{X} = \{X_1, ..., X_n\}$, a Bayesian network B has two components B_s and B_p. B_s is the graphical part, also called the Bayesian network structure which is represented by a directed acyclic graph (DAG). Each node in the graph corresponds to a domain variable. For simplicity, we also name nodes in DAG by $\mathbf{X} = \{X_1, ..., X_n\}$. A directed arc from node X_i to X_j models a dependence relation between X_j and X_i. We say X_i is a parent of X_j. Let Pa_i be the set containing all the parents of X_i. We also use $nonde(X_i)$ to denote all non-descendent nodes of X_i in B_s and

$de(X_i)$ to denote all descendent nodes. Bayesian network has the following important property [25, 19]:

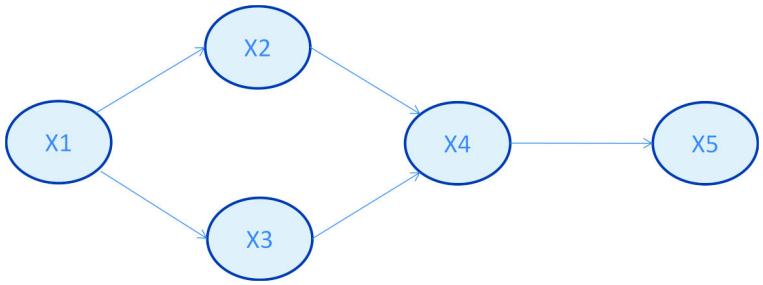

Figure 1.1

An example of Bayesian Network

Theorem 1

(Markov Condition): Every node X_i is independent from $nonde(X_i)$ given Pa_i.

I explain this theorem briefy with a simple network of fve variables shown in Figure 1.1. The Bayesian network has an arc from X_1 to X_2 which indicates that X_1 and X_2 are probabilistically dependent. On the other hand, the absence of an arc between X_1 and X_4 indicates there is no direct dependence between them. According to above theorem, X_1 and X_4 are conditionally independent given X_2 and X_3. Similarly, X_1, X_2 and X_3 are probabilistic independent from X_5 given $Pa_5 = \{X_4\}$. Methods for identifying conditional independence relations in Bayesian networks can be found in [25, 19].

The other component B_p ,which is also called network parameters, provides numerical measurements of the conditional dependence relationships. In a Bayesian network model, each variable is given a conditional probability table in the form of $P(X_i|Pa_i)$. If

X4	State 1	State 2
State 0	0.1	0.3
State 1	0.25	0.4
State 2	0.3	0
State 3	0.25	0.2
State 4	0.1	0.1

Figure 1.2

A conditional probability table.

Pa_i is empty, it degenerates into a prior probability distribution $P(X_i)$. In a Bayesian network, random variables can be either continuous or discrete. For continuous variables, we can use probability density functions to represent the conditional relations. For discrete variables, tables containing all conditional probabilities are used. Figure 1.2 is an example of conditional probability table of X_4 conditional on X_5. In this table, $P(X_4 = state_0|X_5 = state_0) = 0.1$ and $P(X_4 = state_1|X_5 = state_0) = 0.25$. The size of the table is equal to the product of the cardinalities of X_4 and X_5.

With a complete def nition, Bayesian network can now be used to do inference, such as to calculate the joint probability of variables. In the future discussions, I use uppercase letters to represent variables and lowercase letters to represent *instantiations* of variables. If an instantiation involves a set of variables, it means the Cartesian product of the instantiation of each individual variable. Here we use π_i to denote the instantiation of the variable

set Pa_i. Then a formula of calculating the joint probability of n variables can be written as

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i|\pi_i). \tag{1.1}$$

This formula, also called chain rule, is defined to calculate joint probability of all n variables. A more detailed discussion of chain rule can be found in [25]. Using this formula, it is not hard to calculate the probability of any set of variables by eliminating other irrelevant variables. Let $\mathbf{Y}$, $\mathbf{Z}$ be two subsets of $\mathbf{X}$. The formula of calculating their probabilities can be written as:

$$P(\mathbf{Y}) = \sum_{X_i \in \mathbf{X}, X_i \notin \mathbf{Y}} P(\mathbf{X}) \tag{1.2}$$

Also we can compute $P(\mathbf{Y}|\mathbf{Z})$ by using Bayesian rule

$$P(\mathbf{Y}|\mathbf{Z}) = P(\mathbf{Y}, \mathbf{Z})/P(\mathbf{Z}). \tag{1.3}$$

Besides using this brute-force approach above to calculate probabilities, there are many other more efficient algorithms [10, 18, 28, 29].

1.2 Learning Bayesian networks

In the last section, we discussed that a complete Bayesian network contains B_s and B_p. Correspondingly, learning Bayesian networks also contains two tasks. One is to estimate the parameters B_p. Some popular algorithms for learning parameters can be found in [25]. The other is to find a graphic structure B_s which best fits the data. There are many popular algorithms for learning Bayesian networks from data. Generally, they are divided into two categories.

All algorithms falling in the first category are called constraint-based learning algorithms [32, 26, 36, 37]. They assume that data implies independence and conditional independence relationships among variables that can be inferred by using statistical testing or some other non-Bayesian approaches and that there exists a DAG which entails all or at least most of these relationships. One popular statistical testing method can be found in [27]. Using this conditional independence information, we are able to determine the presence and absence of arcs between variables and therefore to build the whole network. There are many variations of this basic idea. One of them is the Greedy Thick Thinning algorithm [4]. It uses conditional independence and dependence relationships obtained from mutual information tests to greedily add and delete arcs between variables. Constraint-based algorithms often require a lot of data in order for the results to be reliable [2], which is often unsatisfied in practice.

Algorithms in the second category are based on Bayesian approaches and are often called score-based methods. They assume a search space which contains all network structures satisfying the directed acyclic constraint and assign a global prior probability to each network. By using Bayesian approaches, we can compute the posterior probability for each network given the data. According to the Maximum Likelihood principle in statistics, a network which has the largest posterior probability best fits the data and is therefore optimal in our consideration. However, sometimes we have many other concerns, such as the complexity of a DAG, overfitting, etc. These considerations have been formulated as various scoring principles, which I will discuss later. Some of them give better networks

higher scores, while others give better networks lower scores. No matter what scoring criteria we choose, the purpose is to find a network to achieve the optimal score.

1.3 Thesis Outline

This thesis proposes a new algorithm based on heuristic search for learning optimal Bayesian network structures from data. We do not consider learning model parameters in this research. For convenience, we assume that the terms *learning Bayesian network* and *learning Bayesian network structure* have the same meaning and can be used equivalently. As we discussed before, there are two types of learning algorithms. Our algorithm falls in the second group. The key idea of this algorithm is applicable to any scoring principle if a good heuristic can be found.

The remainder of the thesis is structured as followings. Chapter 2 formulates the task of learning Bayesian network and then reviews three popular algorithms for solving it. Chapter 3 discusses a new learning algorithm which is based on heuristic search. Chapter 4 provides empirical results for evaluating the efficiency of this new algorithm. Chapter 5 summarizes the contribution of this thesis.

CHAPTER 2

LEARNING BAYESIAN NETWORK FROM DATA

2.1 Basic Assumptions

Before discussing learning Bayesian network, I make three basic assumptions. These assumptions that help to simplify the learning task are satisfied in most practical problems. These assumptions come from [8].

2.1.1 Assumption 1. All variables in the database are discrete

This assumption requires all variables to be discrete and have finite number of instantiations. For continuous variables, there are many effective methods to transform them into discrete ones [22, 13, 12]. With this assumption, all conditional probabilities of a variable given its parents can be stored in a table. Otherwise, a continuous probability density function is needed, which usually makes the learning task difficult. Currently, there are just a few existing algorithms that handle continuous variables and they typically assume that the probability density function follows the normal distribution. More about these algorithms can be found in [25].

2.1.2 Assumption 2. Data cases are independent given a Bayesian network.

A database is a list which contains N cases. Each case is an instantiation of a set of variables. With this assumption, the database is def ned to be a random sample in which each case happens independently. A simple example is the coin toss experiment. In each trial, the probability with which a coin lands head or tail is the same and is not inf uenced by the result of any previous or future trial.

2.1.3 Assumption 3. There are no missing data in database.

No missing data means each case in our database is a complete instantiation of all the variables of the domain. This assumption enables us to ignore data interpolation step which is important in data mining. Although many technical diff culties in practice may inevitably bring in missing values for some variables, there are many effective algorithms available for us to f ll in missing values before the learning process. Some of these techniques can be found in [12, 13, 22, 15]. Figure 2.1 is an example of a database with no missing data.

Cases	Variable_1	Variable_2	Variable_3	Variable_4
1	1	1	0	1
2	1	0	1	1
3	0	1	1	1
4	1	0	1	0
5	0	1	0	0
6	0	1	0	0
7	0	1	0	0

Figure 2.1

Database of four variables

2.2 Basic Model and Data Structure

In a learning problem, we usually rewrite B_s, namely the DAG, as a variable set $\mathbf{X} = \{X_1, ..., X_n\}$ and a set containing their parent sets $\{Pa_1, Pa_2, ..., Pa_n\}$. For instance, Figure 1.1 shows a simple B_s which can rewritten as a pair of sets $\{X_1, X_2, X_3, X_4, X_5\}$ and $\{\{\}, \{X_1\}, \{X_1\}, \{X_2, X_3\}, \{X_4\}\}$. From this example, the parent set of X_1 is empty. The parent set of X_2 is $\{X_1\}$. The parent set of X_3 is also $\{X_1\}$, etc. Now the task of learning optimal Bayesian networks is equivalent to the task of finding a parent set for each variable such that the whole graph is an optimal directed acyclic graph. Now I will introduce several data structures which help to illustrate the learning algorithms.

2.2.1 Parent Graph

Among all possible DAGs with n variables, we let Pas_i to be a set which contains all possible parent sets of variable X_i. The elements in Pas_i are all subsets of $\mathbf{X} - X_i$. For example, assuming that there are three variables in total, X_1 cannot be a parent of itself. Otherwise, there is a direct circle from X_i to itself in the network. Either X_2 or X_3 or both of them can be the parents of X_1. In this case, Pas_1 is $\{\{\}, \{X_2\}, \{X_3\}, \{X_2, X_3\}\}$. Similarly, Pas_2 is $\{\{\}, \{X_1\}, \{X_3\}, \{X_1, X_3\}\}$ and Pas_3 is $\{\{\}, \{X_1\}, \{X_2\}, \{X_1, X_2\}\}$.

Parent Graph is a simple data structure used to store Pas_i for each variable. Figure 2.2 is a parent graph for the variable X_i in which the variable names are represented by the indices. Parent graph in fact is a tree in which each node represents a possible parent set. Each node has one more variable than nodes located in its previous layer. The number

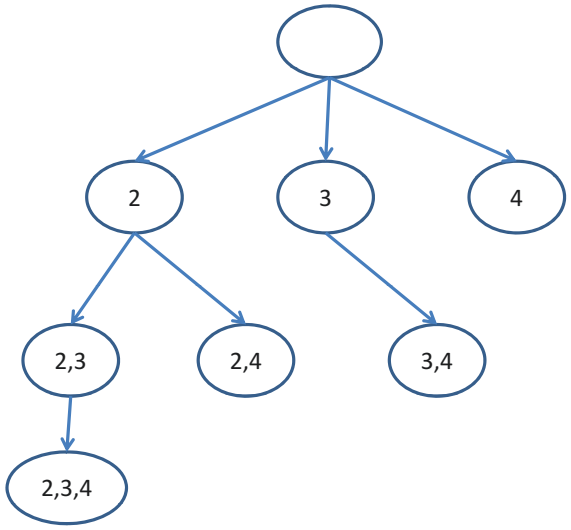

Figure 2.2

All parent sets of variable 1 in set {2, 3, 4}.

of nodes in each layer is equal to C_{i-1}^{n-1} where $(i = 1, 2, ...)$ is the layer index. The total number of nodes is $2^{(n-1)}$.

2.2.2 Order Graph

As described above, we use Pas_i to denote all possible parents of X_i. However, the space formed by the Cartesian product $\prod Pas_i$ contains many cyclic networks. For example, $\{\{X_2\}, \{X_1\}, \{X_2\}\}$ is not a correct B_s because X_1's parent is X_2 and X_2's parent is X_1 which form a circle between these two variables. Some circles will involve three variables or more. These illegal networks should be removed.

In order to describe the space which only contains all legal DAGs, we def ne a total ordering over the variables. This concept enables us to consider each variable's parents independently when constructing DAGs. A total order can be written as $\delta = X_{i_1} \prec X_{i_2} \prec$

$X_{i_3} \prec ... \prec X_{i_n}$ where $X_i \prec X_j$ means X_i is before X_j in the order. There are totally $n!$ possible orders over n variables. Given an order, each variable's parents must be a subset of those variables before it in the order. For example, X_i is a parent of X_j only if $X_i \prec ... \prec X_j$. Using this method to build Pas_i for each variable and construct network space by their Cartesian product, each B_s is a legal directed acyclic graph. This is true because each variable just chooses parents from variables before it, which never produces circles.

With the definition of total ordering, the size of the network space that is consistent with an order can be computed. The number of possible parents for the first variable is $1 = 2^0$ (no parents), for the second is $2 = 2^1$ (no parents or the first variable to be its parent), for the third is $4 = 2^2$, for the ith is $2^{(i-1)}$... As a result, the final size of PB_s is the product of these numbers which is equal to $2^{(n*(n-1)/2)}$.

Order graph is a structure which stores all possible orders over the variables. Figure 2.3 is an order graph for four variables. we usually called the bottom-most node that contains no variable *root* and the top-most node that contains all variables *leaf*. Each node contains several variables indexed by numbers. Each edge connects two nodes one of which contains one more variable than the other. Each path from the root to the leaf which passes n nodes is a total ordering over n variables. The earlier a variable appears in the path, the earlier this variable appears in the ordering. In a dynamic programming algorithm, each node is assumed to be an optimal Bayesian network containing all the variables in the node. Larger networks can be obtained by adding variables. Dynamic programming provides a strategy of finding an optimal Bayesian network using optimal

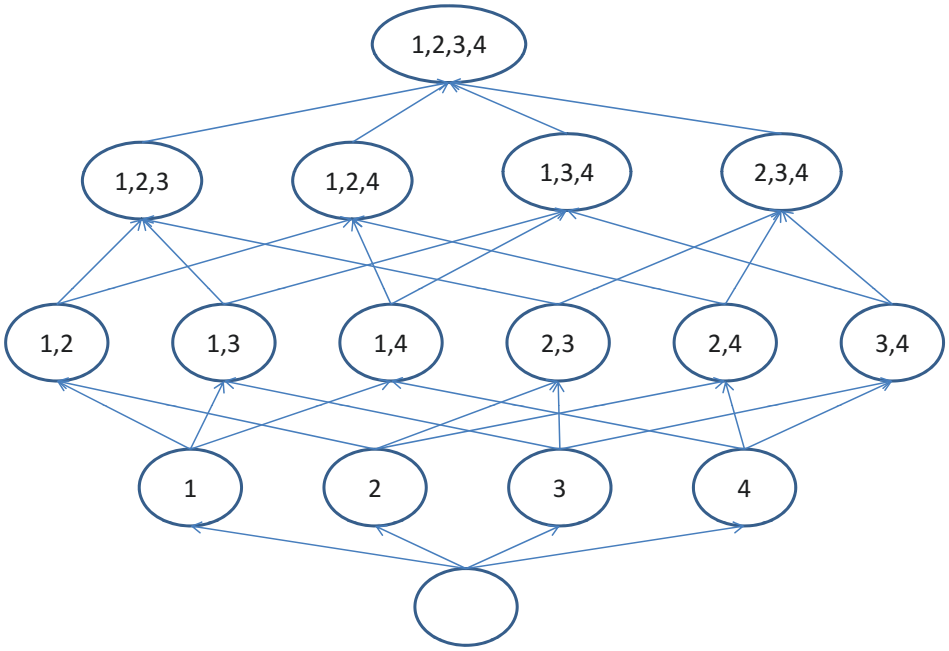

Figure 2.3

Order graph of four variables

networks with one fewer variable. Following this manner, f nally, an optimal network of n variables is obtained. The detailed algorithm will be discussed later.

2.2.3 ADTree

Unlike parent graph and order graph which help to def ne DAG, *ADTree* is a data structure used to count the number of cases in a database that match certain instantiation of variables [23]. These count statistics are utilized by various scoring principles. An ADTree is an unbalanced tree which contains two types of nodes: *varying node* and *ADTree node*. An ADtree node stores the number of cases consistent with the variables instantiation of this node; a varying node is used to instantiate a variable. A full ADTree stores counts of

cases that are consistent with all possible partial instantiations of the variables. Figure 2.4 is an ADTree with n variables and each variable has n_i instantiations. A variable being equal to a star means that this variable can be instantiated by any value when counting matching cases in database. For example, in the root ADTree node, every variable is equal to star, so all the cases in the database matches this node.

Now assume that we want to calculate the count of cases matching query $\{X_1 = *, X_2 = 1, X_3 = *, X_4 = 2\}$. This question can be answered by finding an ADTree node which has exactly the same configuration of these four variables. First, we go to the branch with the varying node X_2 and then go to the ADTree node with $X_2 = 1$. Next, we go to the varying node with X_4 and then ADTree node with $X_4 = 2$. The ADTree node thus found gives the correct count. From this example, it is clear that we branch in an ADTree according to specific values of variables. As a result, we are able to compute the counts of all instantiations of the variable $X_{i_1}, X_{i_2}, ...X_{i_n}$.

2.3 Scoring Metrics

After defining the task of the learning Bayesian network and relevant data structures, we need some rules to measure which network structure fit the observed data better. In this section, I will present two popular scoring principles.

2.3.1 Bayesian Dirichlet (BD) and BDeu

Bayesian Dirichlet (BD) score [8] measures the fitness of B_s to data based on probability. This method assumes that there is a prior probability $P(B_s)$ associated with each

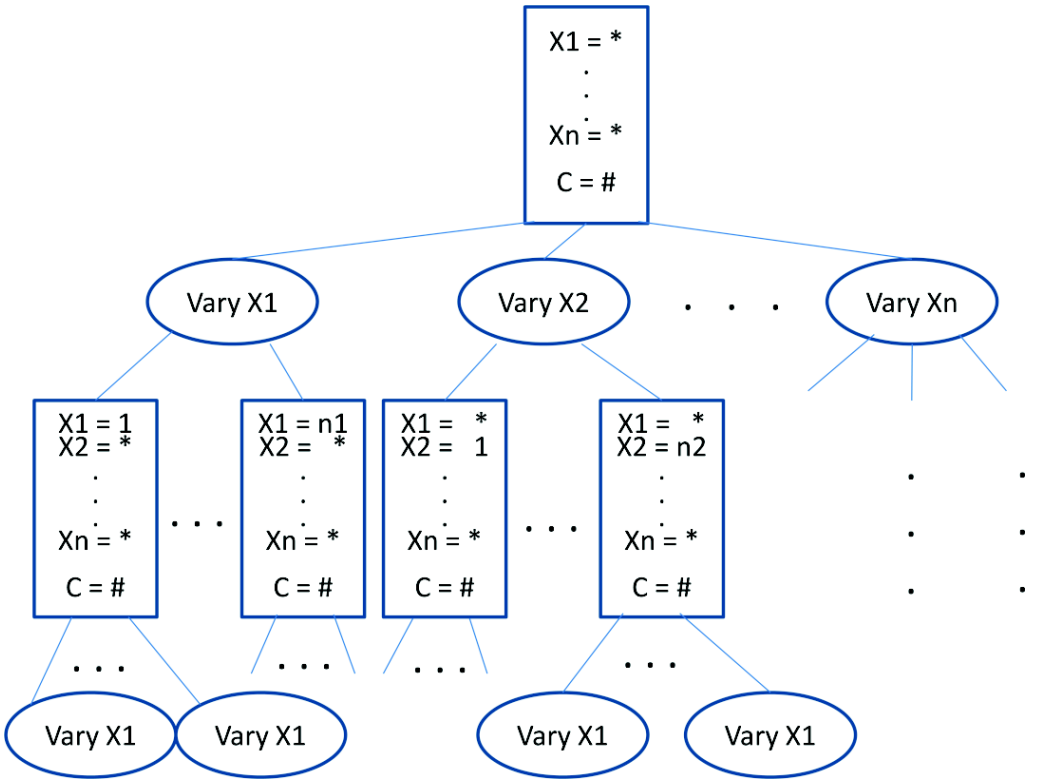

Figure 2.4

Order graph of four variables

legal B_s. Along with each B_s, we use B_p to represent all relevant parameters which help to def ne conditional probability tables and use $f(B_p|B_s)$ to denote the probability of B_p given B_s. Let D be the database which contains a number of cases where D_i is the ith case and D_{ij} be the value of the jth variable in the ith case. Then the BD score can be calculated using formula:

$$P(B_s|D) = \frac{P(B_s, D)}{\int_{B_s} P(B_s, D) dB_s} \propto P(B_s, D) \tag{2.1}$$

$$P(B_s, D) = \int_{B_p} P(D|B_p, B_s) f(B_p|B_s) P(B_s) d_{B_p} \tag{2.2}$$

Since we have assumption 2.1.2 that each case is independent given B_s and B_p, we can rewrite $P(B_s, D)$ to be

$$P(B_s, D) = \int (\prod P(D_i|B_p, B_s)) \cdot f(B_p|B_s) \cdot P(B_s) d_{B_p}. \tag{2.3}$$

Now I will explain each term in the formula. The term $P(D_i|B_p, B_s)$ is easy to obtain using Formula 1.1(the chain rule).

Then it comes to the term $f(B_p|B_s)$. Since each variable is assumed to be discrete, there is a conditional probability table associated with each variable. For variable X_i, let the number of instantiations be r_i and the size of its parents Pa_i be q_i. Also let x_{ik} be the kth instantiation of X_i where $k = 1, 2, ..., r_i$ and π_{ij} be the jth instantiation of Pa_i where $j = 1, 2..., q_i$. Then we use θ_{ijk} to denote $P(x_{ik}|\pi_{ij})$ where $j = 1, 2...q_i$ and $k = 1, 2, ...r_i$. The number of distinct θ_{ijk} equals to $r_i \cdot q_i$. At last, it is reasonable to make following three assumptions.

- θ_{ijk} is independent from $\theta_{ijk'}$
- θ_{ijk} is independent from $\theta_{i'jk}$
- $(\theta_{i1k} + \theta_{i2k}...\theta_{ir_ik}) = 1$

Based on these conditions, $\prod P(B_p|B_s)$ can be decomposed into:

$$\prod P(B_p|B_s) = \Pi_i \Pi_{j=1}^{q_i} f(\theta_{i1k}, ..., \theta_{ir_ik}|Pa_i) \tag{2.4}$$

Now it comes to the key idea of BD scoring principle. It assumes a Dirichlet joint distribution for $\theta_{i1k}, ..., \theta_{ir_ik}$ which are probabilities of X_i given its parents f xed in the kth instantiation. The Dirichlet distribution is a multivariate case of the Beta distribution with which we can write as:

$$f(\theta_{i1k}, ...\theta_{ir_ik};\ \ _{i1k}, ...\ \ _{ir_ik}|Pa_i) = \frac{\prod_{j=1}^{r_i} \Gamma(\ \ _{ijk})}{\Gamma(\Sigma_{j=1}^{r_i}\ \ _{ijk})} \Pi_{j=1}^{r_i} \theta_{ijk}^{\ \ _{ijk}} \tag{2.5}$$

where $i = 1, ..., n$, $j = 1, ..., q_i$, $k = 1, ..., r_i$ and $\ \ _{i1k}, ...,\ \ _{ir_ik}$ are parameters of the distribution.

After f guring out $P(D_i|B_p, B_s)$ and $f(B_p|B_s)$, Cooper and Herskovits [8] then derive the Bayesian Dirichlet score function of B_s against data D. Before giving their results, we f rst def ne the following notations.

- let N be the number of records in D
- let N_{ij} be the number of records in D which match jth instantiation of Pa_i
- let N_{ijk} be the number of records in D which match both kth instantiation of X_i and jth instantiation of Pa_i

With $P(B_s)$ unchanged, the BD score function can be written as [8, 31]:

$$P(Bs, D) = P(Bs) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\ \ _{ij})}{\Gamma(\ \ _{ij} + N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\ \ _{ijk} + N_{ijk})}{\Gamma(\ \ _{ijk})} \tag{2.6}$$

where $_{ijk}$ and $_{ij}$ are parameters. The detailed proof of this result can be found in [8].

In paper [3], the author gives the likelihood equivalent uniform Bayesian Dirichilet (BDeu) score by further making the following assumptions based on BD score:

- $P(B_s)$ is uniformly distributed,
- $_{ij} = \frac{1}{q_i}$
- $_{ijk} = \frac{_{ij}}{r_i}$

Therefore BDeu score can be written as:

$$BDeu(Bs, D) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} BDeu_{ij} \tag{2.7}$$

$$BDeu_{ij} = \frac{\Gamma(\ _{ij})}{\Gamma(\ _{ij} + N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\ _{ijk} + N_{ijk})}{\Gamma(\ _{ijk})} \tag{2.8}$$

Equation 2.8 provides a method to calculate score for X_i with parents fixed to the jth instantiation. That means we can compute the score for each variable and its parents without worrying about other variables. Once the parents of all variables are compatible, the whole structure's BDeu score can be calculated by simply multiplying them together. A scoring principle with such a property is called *decomposable*. There are many scoring principles which possess this property. Decomposability makes the learning task easier.

2.3.2 Minimum Description Length (MDL)

MDL scoring principle is based on information theory [2]. In information theory, information entropy can help us measure the amount of information that is missing upon reception [9]. The formula to compute information entropy of random variable X_i is defined as:

$$H(X_i) = -\sum p(x_{ij}) \log_2 p(x_{ij}) \tag{2.9}$$

This formula measures how much information we need in order to correctly describe variable X_i. In machine learning, Occam's razor provides a good heuristic for selecting a model to fit data. Shortly, Occam's razor can be interpreted as: *a simple model is better; a smoother model is better; and a model with fewer parameters is better.*

The MDL principle uses Occam's razor as heuristic to select a Bayesian network that can describe the data as accurately as possible but with as few parameters as possible. In [2], the MDL principle is proved to be an approximation of the BD principle. Moreover, it offers some more advantages. Here I just give the formula of MDL and explain briefly the meaning of each term in the formula. More details can be found in paper [2].

Definition 1

Let $B_s, D, N, n, q_i, r_i, N_{ijk}, N_{ij}$ be defined as before. The description length MDL(B_s, D) of Bayesian Network structure B_s with respect to D is defined by

$$MDL(B_s, D) = \log P(B_s) - N \cdot H(B_s, D) - (K \cdot \log N)/2 \tag{2.10}$$

where $K = \sum_{i=1}^{n} q_i \cdot (r_i - 1)$ and $H(B_s, D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} -\frac{N_{ijk}}{N} \log \frac{N_{ijk}}{N_{ij}}$

This description length is also MDL score.

The first component $P(B_s)$ is the prior probability of B_s. If we assume all structures have the same prior probabilities, this term can be ignored while comparing different networks.

The second component $-N \cdot H(B_s, D)$ is the conditional entropy of B_s. Since the ratio within the $\log$ is always smaller than 1, this entropy is non-negative. It measures the uncertainty of the model with respect to the data. The larger the entropy is, the more

uncertainty the model possesses. A zero value means that there is no uncertainty at all or that the model is deterministic.

The third component is $(K \cdot \log N)/2$. K is the total number of parameters used to correctly describe all conditional probability tables. For example, for X_i and each instantiation π_{ij} of parents, we need $(r_i - 1)$ parameters $\theta_{ij0}, ...\theta_{ij(r_i-1)}$. The remaining one θ_{ijr_i} is automatically calculated since the sum of all probabilities is equal to one. Thus, the total number of parameters of X_i is $q_i(r_i - 1)$. As we know in statistics, estimation of parameters will definitely bring errors. Intuitively, the number of parameters is proportional to the amounts of errors introduced by estimation. Bouckaert in his paper [2] says $(K \cdot \log N)/2$ measures number of parameters in the model and therefore is also a measurement of the error introduced by estimating all required probabilities.

In total, because of the last two terms, the MDL score provides a measurement that takes into consideration a model's simplicity and its goodness of fit to the data [33]. In this principle, the smaller the MDL score is, the better a model is. Thus, our goal is to find a model that minimizes the MDL score.

2.4 Approximate Algorithm

With the definitions of Bayesian network and scoring principles, in this section, I briefly review several approximate learning algorithms. It has been shown that structure learning is NP-hard [7]. Given n variables, there are $O(n2^{n(n-1)})$ directed acyclic graphs (DAGs). Since the size of the solution space grows super-exponentially in the number of variables, early research focused mainly on approximate algorithms. Various local search

methods have been proposed to search for high-scoring structures, including tabu search, restarting, and simulated annealing [2, 5, 17].

Several other methods improve simple local search using different strategies, including ordered variables [8, 34], bound on number of parents [20, 21, 14], greedy equivalent search that searches the space of equivalence classes [6], and optimal reinsertion that greedily applies an optimal reinsertion transformation repeatedly on the graph [24]. Yet some other methods combine constraint-based learning [32] with local search for finding Bayesian network structures [35].

2.5 Dynamic Programming Algorithm

In recent years, researchers began to study how to find optimal network structures based on dynamic programming. In this section, I review a popular dynamic programming algorithm [31]. This algorithm can find an optimal Bayesian network structure in $O(n2^n)$ time. Another similar algorithm with the same worst time complexity is presented in paper [30].

Singh and Moore [31] developed a dynamic programming algorithm based on BDeu. In fact, other scoring principles such as MDL are also applicable. The basic idea is that each DAG graph contains at least one leaf node with no children nodes. Here let us assume there are n nodes. Each Bayesian network consists of two parts: a leaf node X_i and a subnetwork in which X_i and all arcs connected to X_i are removed. Since the BDeu score is decomposable,we can compute scores independently for each node given its parent set. Therefore, if a network structure is optimal, both the leaf node and the subnetwork should

be optimal. The leaf node X_i is optimal in the sense that it has an optimal parent set Pa_i out of the remaining variables such that the node score $NScore(X_i|Pa_i)$ is maximized (remember that all remaining nodes can be its parents). The subnetwork is optimal in the sense that it is the best network for the remaining $n-1$ variables. More formally, the score of an optimal network for variables $\mathbf{V}$, $Score(\mathbf{V})$, can be expressed as the sum of two parts[31].

$$Score(\mathbf{V}) = \max_{X_i \in \mathbf{V}} Score(\mathbf{V} \setminus \{X_i\}) + BestScore(\mathbf{V}, X_i), \tag{2.11}$$

where

$$BestScore(\mathbf{V}, X_i) = \max_{Pa_i \subseteq \mathbf{V} \setminus \{X_i\}} NScore(X_i|Pa_i). \tag{2.12}$$

Figure 2.5 gives the pseudocode of dynamic programming algorithm. In the pseudocode, we use Net_V to denote an optimal subnetwork consisting of $\mathbf{V} = \{V_1, V_2, ..., V_m\}$. Our goal is to find an optimal network Net_X which consists of all variables denoted by $\mathbf{X} = \{X_1, X_2, ..., X_n\}$.

This algorithm can be well illustrated by the order graph. Figure 2.3 shows an order graph used by dynamic programming to find an optimal Bayesian network with four variables. In this graph, each node represents an optimal subnetwork consisting of the variables in it. For example, $\emptyset$ represents an empty subnetwork which is trivially optimal. Nodes in the second layer represent all optimal subnetworks containing one variable. Based on the second layer, each node in the third layer can be built which represents an optimal subnetwork with two variables. Proceeding in this manner, the optimal network with four variables is obtained at the top layer. This layer just has one node which contains

Algorithm: Learning optimal Bayesian network using Dynamic Programming;
Input: a dataset with n variables $\mathbf{X} = \{X_1, X_2, ...X_n\}$;
Output: an optimal Bayesian network.

1. Find all optimal structures containing just single variable.
2. **for** k = 2,..., n
3. **for** each k variable set $\mathbf{V} = \{X_{i1}, ...X_{ik}\}$
4. **for** j = 1 to k
5. Find for X_{ij} an optimal parent set from $\mathbf{V} \setminus \{X_{ij}\}$.
6. Let $Net_{V_{-ij}}$ be an optimal network for$V_{-ij} = \mathbf{V} \setminus \{X_{ij}\}$.
7. Construct Net_{V_j} by adding X_{ij} into $Net_{V_{-ij}}$.
8. **end for**
9. Let Net_V be the optimal one from all the Net_{V_j}s.
10. Store this optimal network and its score.
11. **end for**
12. **end for**
13. Output the optimal network of Net_X where $\mathbf{X} = \{X_1, X_2, ..., X_n\}$.

Figure 2.5

Dynamic Programming for learning optimal Bayesian networks.

all n variables. For the node $\{1, 3, 4\}$, there are three nodes or optimal subnetworks in the third layer having an arc pointing to it. This means the optimal subnetwork of $\{1, 3, 4\}$ can be obtained by adding variable to one of these three optimal subnetworks.

Also, as shown in Figure 2.3, each directed arc means a problem of finding the optimal parents from a set of variables called *candidate set*. For example, the arc from node $\{1, 4\}$ to node $\{1, 3, 4\}$ means to find the optimal parents for variable 3 from the candidate set $\{1, 4\}$. These operation can be finished using parent graph as it links all possible parent sets into a tree structure. This data structure makes it efficient to search the optimal parent set which is consistent with a candidate set.

Dynamic programming finds optimal subnetworks for all nodes in the order graph. It first chooses a variable and decomposes the original network into the variable and a subnetwork with it removed. It then computes the total score of network. Then algorithm chooses another variable to decompose the original network and calculates its score again. In this manner, algorithm considers each variable once and selects the one with the highest score. Finally, the algorithm constructs an optimal network by combining the selected variable and its parents with the optimal subnetwork with this variable removed.

2.6 Systematic Search Algorithm

A recent systematic search method uses theoretical properties of the MDL score to reduce the size of solution space [11]. It is found that some parent sets are guaranteed to be suboptimal without computing their scores. In particular, it is shown that, in an optimal Bayesian network, each node can have at most $O(\log N)$ parents where N is the number

of data points [11, 34]. Due to limited number of data points in a typical dataset, it is easy to prune the parent space (def ned above) and therefore decrease the computation to f nd optimal parent set.

The search search algorithm f rst calculates the scores for all the valid parent sets. It then f nds optimal parent sets for all the variables by initially ignoring the acyclic constraint. The result is a directed graph with cycles. The algorithm then repeatedly f nds directed cycles in the candidate graph and systematically goes through all cycle-breaking strategies by removing one arc at a time. After breaking all circles, we obtain an optimal Bayesian network.

CHAPTER 3

A HEURISTIC SEARCH ALGORITHM

3.1 Basic Motivation

From the algorithm in Figure 2.5, it is easy to see that dynamic programming finds optimal subnetworks for all subsets of X. which in turn requires computing all the parent scores of each variable. For n variables, there are 2^n nodes to evaluate in the order graph. For each node, there is a parent graph which stores 2^{n-1} parent sets along with their scores. The total number of parent sets is $n \cdot 2^{n-1}$. This number grows very fast with respect to n. The algorithm[31] computes and stores all these parent sets. As a result, while the number of variables increases, the algorithm becomes infeasible as it requires too much memory and too much time to compute and store all parent sets.

The systematic search algorithm in [11] was able to reduce the computation using the theoretical properties. However, it was shown to be less efficient than dynamic programming on some datasets. We believe there are two major reasons. First, it initially ignores the acyclic constraint and finds optimal parent sets independently for all the variables. The number of parent scores that need to be computed can still be prohibitive and the search space containing cyclic networks is huge as well. Second, it needs to repeatedly detect and break cycles in directed graphs, which may be expensive for large graphs and results in an inefficient search.

In this chapter, I present an improved search method for finding optimal Bayesian networks to address the drawbacks of dynamic programming and systematic search. We first introduce our formulation and related techniques for improving search efficiency and then end this chapter with a pseudocode of the algorithm and a discussion on its advantages.

3.2 Formulation

The basic idea of our algorithm is to formulate learning optimal Bayesian networks as a *shortest path finding* problem. We use the order graph as the search graph. We view the root of order graph as the *start state* and the leaf as the goal state.

For any two neighboring nodes $\mathbf{S_1}$ and $\mathbf{S_2}$ with an arc from $\mathbf{S_1}$ to $\mathbf{S_2}$, We define the *edge cost* $c(\mathbf{S_1}, \mathbf{S_2})$ to be $-BestScore(\mathbf{S_2}, X_i)$, where X_i is the only variable the two nodes differ.

Because we use the MDL score, we set the edge cost to be $MDL(X_i|Pa_i)$, where Pa_i is an optimal parent set for X_i out of $\mathbf{S_1}$. The goal is then to find a *shortest path* from the start state to the goal state that has the minimal cost. By definition, the shortest path corresponds to a Bayesian network with the maximum total score.

Once we formulate the problem as a shortest path finding problem, we can apply any graph search technique to solve it. In this thesis, I present a *best-first heuristic search* algorithm, i.e., an *A* search* algorithm. We use a priority queue, called OPEN list, to organize the search frontier and initialize it with the start state. At each search step, we pop up the search node with the smallest cost from the OPEN list and expand its children search nodes. For each search node, we compute its f cost, the estimated total cost, as the

sum of g cost, the exact cost so far, and h cost, the estimated future cost to the goal state. Once a node is expanded, it is placed in a CLOSED list. Duplicate detection is performed for each newly generated node on both OPEN and CLOSED lists. If a duplicate is detected in the CLOSED list, we discard the new node immediately because we show later that we use a consistent heuristic. If a duplicate is detected in the OPEN list and the new node has a lower g-cost, we update the existing node with the new g-cost and parent pointer.

The g cost is computed as the sum of edge costs on the best path from the start state to the current state. Each edge cost is computed when the end node of the edge is generated by the search. The computation is achieved by searching a suitable parent graph. For example, when the edge cost between nodes $\{2, 3\}$ and $\{1, 2, 3\}$ in Figure 2.3 is needed, we will go to the parent graph of variable 1 and search for a subset of $\{2, 3\}$ that has the highest score. Therefore, our method is a *two-layer nested search* algorithm. The higher-level search works on the order graph and f nds a shortest path. Whenever the higher-level search needs an edge cost, a lower-level search is deployed to f nd an appropriate score on a parent graph.

Since the A* search only explores part of the order graph, we only need to compute some of the edge costs. This pruning is inherent in the search algorithm and is not reliant on any property of the scoring function.

If we use a h function that is not only *admissible* but also *consistent*, the A* search algorithm guarantees to f nd a shortest path once the goal state is selected for expansion. We can extract an optimal Bayesian network out of the shortest path as each edge on the

path records an optimal parent set for a variable. Let $\mathbf{U}$ be a node on the order graph. We consider the following h function.

Def nition 2

$$h(\mathbf{U}) = -\sum_{X_i \in \mathbf{V} \setminus \mathbf{U}} BestScore(\mathbf{V}, X_i). \tag{3.1}$$

h is clearly admissible, i.e., it always underestimate the cost (or equivalently, overestimate the score). h allows each remaining variable to select optimal parents from all the other variables in $\mathbf{V}$. This effectively relaxes the acyclic assumption and results in a lower bound cost. The following theorem proves that the heuristic is also consistent. A consistent heuristic is guaranteed to be admissible.

Theorem 2

h is consistent.

Proof: For any successor node $\mathbf{R}$ of $\mathbf{U}$, let $Y \in \mathbf{R} \setminus \mathbf{U}$. We have

$$\begin{aligned} h\mathbf{U}) &= -\sum_{X_i \in \mathbf{V} \setminus \mathbf{U}} BestScore(\mathbf{V}, X_i) \\ &\leq -\sum_{X_i \in \mathbf{V} \setminus \mathbf{U}, X_i \neq Y} BestScore(\mathbf{V}, X_i) \\ &\qquad -BestScore(\mathbf{R}, Y) \\ &= h(\mathbf{R}) + c(\mathbf{U}, \mathbf{R}). \end{aligned}$$

The inequality holds because fewer variables are used to select optimal parents for Y. Hence, h is consistent. ■

3.3 Parent graph pruning

Parent graphs are used to compute both g and h costs in our search algorithm. The h-cost seems expensive to compute because it requires computing the optimal scores $BestScore(\mathbf{V}, Y_i)$ for all remaining variables. This is equivalent to finding an optimal parent set for each variable out of all the other variables, which requires a complete search on the variable's parent graph. However, we only need to compute these best scores once in the beginning of the search. The scores are repeatedly used in later search. It is more expensive to compute the g-costs, which amounts to compute the edge costs. Computing each edge cost requires searching a parent graph to find the highest score among subsets of given candidate parents.

Smaller parent graphs clearly will make computing g and h costs more efficient. We utilize several pruning techniques to reduce the size of parent graphs.

One technique relies on the following theorem presented in [11, 34] to prune large parent graphs.

Theorem 3

In an optimal Bayesian network based on the MDL scoring function, each variable has at most $\log(\frac{2N}{\log N})$ parents, where N is the number of cases.

Therefore, there is no need to compute scores for any parent set whose size is larger than $\log(\frac{2N}{\log N})$.

Another technique prunes parent sets that are guaranteed to be worse than a common subset parent set without computing their exact scores based on the following theorem presented in [31].

Theorem 4

Let $\mathbf{U} \subset \mathbf{V}$ and $X \in \mathbf{U}$. Let $hScore(X, \mathbf{U}, \mathbf{V})$ be an upper bound which bounds $BestScore(\mathbf{R}, X)$ for any $\mathbf{R}$ such that $\mathbf{U} \subset \mathbf{R} \subseteq \mathbf{V}$. Then if $hScore(X, \mathbf{U}, \mathbf{V}) < BestScore(\mathbf{U}, X)$, no proper superset of $\mathbf{U}$ can be optimal parent set for X.

To use this theorem, we need the upper bound score $hScore(X, \mathbf{U}, \mathbf{V})$. Since we use the MDL score, we use the following lower bound for MDL def ned in [33].

Theorem 5

Let $\mathbf{U} \subset \mathbf{V}$ and X_i be a variable not in $\mathbf{V}$. For any $\mathbf{R}$ such that $\mathbf{U} \subset \mathbf{R} \subseteq \mathbf{V}$, we have

$$MDL(X_i|\mathbf{R}) \geq \frac{\log N}{2} K(X_i|\mathbf{U}). \tag{3.2}$$

We do not use the tighter lower bound for MDL def ned in Theorem 6 presented in [34]. The reason is calculating $H(X_i|\mathbf{V})$ requires that we collect count statistics for full con- f gurations of the variables for a dataset, which is too expensive for large datasets. As we will discuss in the next section, we prune the counts for large variable conf gurations using Theorem 3.

Theorem 6

Let $\mathbf{U} \subset \mathbf{V}$ and X_i be a variable not in $\mathbf{V}$. For any $\mathbf{R}$ such that $\mathbf{U} \subset \mathbf{R} \subseteq \mathbf{V}$, we have

$$MDL(X_i|\mathbf{R}) \geq H(X_i|\mathbf{V}) + \frac{\log N}{2} K(X_i|\mathbf{U}). \tag{3.3}$$

Finally, we also use Theorem 7 [11] to prune some parent scores that are already computed to reduce the sizes of parent graphs.

Theorem 7

Let $\mathbf{U} \subset \mathbf{V}$ and $X \in \mathbf{U}$. If $BestScore(\mathbf{U}, X) > BestScore(\mathbf{V}, X)$, $\mathbf{V}$ cannot be optimal parent set for X.

Potentially we can generate the parent graphs incrementally during the search. That means we only generate the parent scores when they are needed in the search, which seems able to generate smaller parent graphs. It turns out not to be the case. If we generate the graphs incrementally, we have to delay the use of Theorem 7 due to incomplete parent graphs. Experiments indeed show that this method results in larger parent graphs. Therefore, we choose to compute all the parent graphs before the search while using the above pruning techniques to reduce their sizes.

3.4 AD-tree pruning

For n variables with d states each, the number of ADtree nodes in an AD-tree is $(d+1)^n$. It grows even faster than the sizes of order and parent graphs. It is impractical to compute and store all the count statistics for a large dataset. Theorem 3 requires that we only compute scores for small parent sets. Consequently, we only need to collect count statistics for small variable instantiations as well. We can prune large variable instantiations from the AD-tree. We believe this pruning will signif cantly increase the scalability of our search algorithm.

3.5 The A* search algorithm

A pseudo code of our algorithm is shown in Figure 3.1. We first construct an AD-tree for the input dataset and create all the parent graphs. The main body of the algorithm is essentially an A* search algorithm. We extract an optimal Bayesian network out of the shortest path in the end.

The major advantage of our A* search algorithm over dynamic programming is that the A* search only needs to explore part of an order graph and compute some of the edge costs on the graph. In comparison, dynamic programming evaluates the order graph completely and compute all edge costs. It is clear from Figure 2.3 that an order graph is typically densely connected. The pruning by our search algorithm is clearly important for large order graphs.

However, each step of our search algorithm has the overhead of computing the heuristic values, although the computation is much cheaper when compared to computing an edge cost. Therefore, a search step is slightly more expensive than a similar dynamic programming step. If the pruning does not out weigh the overhead, the search algorithm can be slower than dynamic programming. We believe for large datasets, the gain brought by the pruning will significantly out weigh the overhead.

A major difference between our A* search algorithm and the systematic search method is that our algorithm always maintains an acyclic directed graph during the search. There is no need to detect or break cycles in directed graphs. This difference turns out to be a huge advantage for our search algorithm.

Algorithm: Learning optimal Bayesian network;
Input: a dataset with variables **V**;
Output: an optimal Bayesian network for the dataset.

1. Create an AD-tree to collect suff cient statistics from the dataset
2. Create parent graphs for all the variables
3. Initialize OPEN list with the start state
4. **while** OPEN list not empty
5. Remove the best node n from the OPEN list
6. **if** n is the goal node
7. Extract and return a Bayesian network
8. **end if**
9. Put n in the CLOSED list
10. Expand successor nodes of n
11. **for** each successor s
12. **if** s in CLOSED list
13. continue
14. **end if**
15. Compute the edge cost from parent node
16. Compute h-cost to the goal state
17. Compute f-cost
18. **if** s in OPEN list & current g cost is lower
19. Update g cost and parent pointer
20. **else**
21. Add s to OPEN list
22. **end if**
23. **end for**
24. **end while**

Figure 3.1

An A* search algorithm for learning optimal Bayesian networks.

CHAPTER 4

EMPIRICAL RESULTS AND ANALYSIS

4.1 Experiments and Results

In order to evaluate the performance of our algorithm, I test it on a set of benchmark datasets from the UCI repository [1] listed in Table 4.1. The largest datasets have up to 24 variables and $32,561$ data points. I discretized all continuous variables or discrete variables with more than three states into two states and filled in random values for the datasets with missing data.

In the experiments, I compared my algorithm against dynamic programming [30, 31] and systematic search [11]. The two dynamic programming algorithms presented in [30, 31] only differ in that the method in [31] uses Theorem 4 to prune parent graphs. I implemented the version with pruning (denoted as 'DP'). I also note that Theorem 3 and 7 can be applied to the dynamic programming algorithm to improve its time and space efficiency. For fair comparison, I implemented an enhanced dynamic programming algorithm with all the pruning techniques used by our search algorithm (denoted as 'DP-E'). For the systematic search algorithm (denoted as 'SS'), I downloaded the binary code made public by its authors from the following website: http://www.ecse.rpi.edu/$\sim$ cvrl/structlearning.html. This code only allows AIC or BIC scores. I choose to use BIC because it is considered equivalent to MDL.

My experiments were performed on a 3.2 GHz processor with 4 gigabytes of RAM running a 64-bit version of Windows XP.

4.2 Discussion

Table 4.1 is a comparison on the running time and sizes of order and parent graphs for the following algorithms: Dynamic programming with score-bound parent graph pruning (DP), Enhanced dynamic programming algorithm with all parent graph pruning (DP-E), Systematic search (SS), and the A* search algorithm. The column headings have the following meanings: 'n' is the total number of variables; 'N' is the number of cases; 'Time' is the running time in seconds; 'orderNodes' is the number of nodes evaluated by the A* search in order graph; 'f-orderNodes' is the number of nodes evaluated by dynamic programming in order graph; 'orderEdges' is the number of edges evaluated by the A* search in order graph; 'f-orderEdges' is the number of edges evaluated by dynamic programming in order graph; 'adtree' is the size of computed AD-trees; 'f-adtree' is the size of full AD-trees. Dynamic programming evaluates all nodes and edges in order graphs ('f-orderNodes' and 'f-orderEdges'), while our A* search algorithm only generates partial order graphs ('orderNodes','orderEdges'). Although dynamic programming and A* search compute the same AD-trees, I include the sizes of full AD-trees without pruning ('f-adtree') and the actual AD-trees used by the algorithms ('adtree') to emphasize the importance of AD-tree pruning. Finally, '-' shows failure due to time out or running out of memory.

Table 4.1

Experimental statistics of several learning algorithms

Dataset			Timing results				Memory statistics					
dataset	n	N	DP	DP-E	SS	A*	orderNodes	f-orderNodes	orderEdges	f-orderEdges	adtree	f-adtree
iris	5	150	0.02	0.00	0.03	0.01	27	32	33	80	598	1,024
liver	7	345	0.04	0.02	0.12	0.02	123	128	263	448	2181	2,187
car	7	1,728	0.11	0.07	0.16	0.11	114	128	202	448	40,000	40,000
acute	8	120	0.04	0.02	0.36	0.03	210	256	445	1,024	2,942	8,748
abalone	9	4,177	0.24	0.26	52.32	0.28	506	512	1,647	2,304	23,440	34,992
cmc	10	1,473	0.43	0.49	0.83	0.49	961	1,024	2,221	5,120	336,706	1,080,000
wine	14	178	2.97	3.41	26.31	1.45	13,465	16,384	38,627	114,688	504,436	6,377,292
adult	15	32,561	97.87	135.90	-	128.52	32,750	32,768	207,948	245,760	9.015E+06	1.435E+07
credit	16	690	21.42	23.10	196.58	16.23	65,517	65,536	332032	524,288	4.348E+06	1.020E+08
zoo	17	101	30.08	31.41	377.46	8.16	112,820	131,072	365,685	1,114E+06	4.691E+05	1.291E+08
letter	17	20,000	440.1	779.16	-	695.44	130,879	131,072	1.040E+06	1,114E+06	6.869E+07	1.291E+08
voting	17	435	-	26.61	-	12.77	98,177	131,072	263,839	1,114E+06	1.708E+07	1.288E+10
hepatitis	20	155	127.92	91.44	-	30.70	494,599	524,288	1.802E+06	1.049E+07	1.624E+06	3.487E+09
meta	22	528	-	16,171.89	-	944.13	2.083E+06	4.194E+06	8.859E+06	4.614E+07	1.726E+07	1.046E+10
Heart	23	80	-	5,003.64	-	1,137.89	8.034E+06	8.389E+06	4.282E+07	9.647E+07	4.601E+06	9.414E+10
parkinson	24	197	-	26,542.40	-	10,032.36	1.662E+07	1.678E+07	9.791E+07	2.013E+08	2.661E+07	2.824E+11

The timing results show that our algorithm signif cantly improves the eff ciency of learning optimal Bayesian networks for these benchmark datasets. Both DP and SS algorithms ran out of memory on several of the datasets, while DP-E and A* were able to solve all of them. It is not surprising that the additional pruning techniques helped DP-E achieve much better eff ciency than DP. Still, our A* search algorithm signif cantly outperforms the other algorithms on most of the datasets. The speedup ranges from several times faster to orders of magnitude faster (e.g., wine, zoo, meta, heart). Our algorithm is only slower than dynamic programming on several small datasets. It is due to the overhead of computing bounds at each search step, as I explained earlier. The difference seems negligible though. Since big improvement is typically observed on the large datasets, the results indicate that our algorithm scales up better than the other algorithms.

The comparison between the sizes of order graphs shows that our algorithm helps prune some of the nodes and edges. From the table, we found that the reduction of edges can better explain the algorithm improvement than order nodes. This is because, as I explain earlier, the most expensive part of evaluating an order graph is computing the edge costs. For example, for problem “zoo”, the pruning of order nodes is just about 15%, but the reduction of edge is 67%. As a result, the time improvement of A* compared with DP-E reaches 74%. For problem “hepatitis”, order nodes pruning is just about 6% but the edge pruning is 83%. Again, the time improvement is 66.5%. From the above results, we found the reduction for edges is always much larger than that for order nodes. This is because each node has many outgoing edges.

Another interesting result in the table is that the time improvement rate of A* compared with DP-E is always slightly bigger than edges reduction rate. An example is “parkinson”. The edge reduction is about 51% while timing improvement is 62%. And for “Heart”, the edge reduction is about 56% while time improvement is 77%. The reason of this phenomena is that most of the edge pruning happens in deep layers of the order graph that are more expensive to compute. In the beginning of the search, the heuristic value is loose because it is equal to the sum of best scores for many variables. As the search gets deeper, more variables obtain exact scores, which makes the heuristic value become tighter. As a result, the nodes and especially the edges in deep layer are more likely to get pruned. It is desirable to prune these deep edges because their costs are more expensive to compute. The reason for this is that computing the costs amounts to selecting optimal parent sets for the variables. In the beginning layers, there are only few candidate parents to choose from. As the depth becomes large, the number of candidate parents also increases, and the number of possible parent sets grows exponentially in the number of candidate parents. Therefore, the cost of computing an edge cost also grows with the depth. Therefore, even though sometimes not many edges on the order graph are pruned, I pruned the most expensive edges, which results in signif cant improvement in search eff ciency.

The sizes of AD-trees clearly show that it is impossible to compute and store the full AD-trees for large datatsets. I have to minimize the AD-trees using pruning. Otherwise, the algorithms will fail to construct the AD-trees before the search for an optimal structure even starts.

The systematic search algorithm [11] is much slower than our search algorithm. In our experiments, I did not specify any constraints. The results make it evident that the systematic search algorithm is much less efficient than our algorithm. It is better to maintain an acyclic directed graph in searching for an optimal Bayesian network.

CHAPTER 5

CONCLUSION

In this thesis, I describe a new algorithm for learning optimal Bayesian networks based on the best-first heuristic search A* algorithm. It uses a consistent heuristic to guide the search such that only the most promising parts of the solution space is explored. This not only allows an optimal solution to be found much more efficiently, but also in less space. The methods will enable the application of Bayesian learning methods to larger real-world applications. More importantly, the proposed methods can allow modelers to focus on the modeling issues and the interpretation of the learning results without worrying about the quality of the algorithms themselves. The methods will also enable a systematic comparative study of the different choices of priors and scoring functions.

We currently use the MDL score in our algorithm. However, other scoring functions can also be used, as long as they are decomposable, and we also know how to estimate an upper bound for the score. For example, the BDeu score is also a decomposable score. An upper bound for the score is presented in [20]. We can easily adapt our algorithm to find optimal Bayesian network with the highest BDeu score.

Similar to dynamic programming, our algorithm will also benefit from parallel computing and external memory search. One method in consideration to further scale up Bayesian

network learning is to utilize external memory based on the structured duplicate detection technique [25].

REFERENCES

[1] A. Asuncion and D. Newman, "UCI machine learning repository," 2007.

[2] R. R. Bouckaert, *Probabilistic Networks Constuction Using the Minimum Description Length Principle*, Tech. Rep., Utrecht University, The Netherlands, 1994.

[3] W. Bruntine, "Theory ref nement on Bayesian networks," *In Proceedings of Uncertainty in Artificial Intelligence*, 1991.

[4] J. Cheng, D. Bell, and W. Liu, "An algorithm for Bayesian belief network construction from data," *In Proceedings of AI & STAT'97*, 1997.

[5] D. M. Chickering, "A transformational characterization of equivalent Bayesian network structures," *In Proceedings of the 11th annual conference on uncertainty in artificial intelligence*, 1995.

[6] D. M. Chickering, "Learning equivalence classes of Bayesian-network structures," *Journal of Machine Learning Research*, vol. 2, 2002, pp. 445–498.

[7] D. M. Chickering and C. Meek, "Large-Sample Learning of Bayesian Networks is NP-Hard," *In Proceedings of Uncertainty in Artificial Intelligence 2003*, 2003.

[8] G. F. Cooper and E. Herskovits, "A Bayesian Method for the Induction of Probabilistic Networks from Data," *Machine Learning*, vol. 9, 1992, pp. 309–347.

[9] T. M. Cover and J. A. Thomas, *Elements of Information Theory, Second Edition*, Wiley, 2006.

[10] P. Dawid, "Applications of a general propagation algorithm for probabilistic expert systmes," *Statistics and Computing*, vol. 2, 1992, pp. 25–36.

[11] C. de Campos, Z. Zeng, and Q. Ji., "Structure learning of Bayesian networks using constraints," *In Proceedings of the International Conference on Machine Learning*, 2009.

[12] I. Ebert-Uphoff, *A Probability-Based Approach to Soft Discretization for Bayesian Networks*, Tech. Rep., Georgia Institute of Technology. School of Mechanical Engineering, 2009.

[13] N. Friedman and M. Goldszmidt, "Discretizing Continuous Attributes While Learning Bayesian Networks," *In Proc. ICML*, 1996.

[14] N. Friedman, I. Nachman, and D. Peer, "Learning Bayesian network structure from massive datasets: The sparse candidate algorithm," *In Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, 1999.

[15] J. Gama, L. Torgo, and C. Soares, "Dynamic Discretization of Continuous Attributes," *In Proceedings of the Sixth Ibero-American Conference on AI*, 1998.

[16] D. Heckerman, *A tutorial on learning with Bayesian networks*, Redmond, Washington, 1996.

[17] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," *Machine Learning*, vol. 20, 1995, pp. 197–243.

[18] F. Jensen, S. Lauritzen, and K. Olesen, "Bayesian updating in recursive graphical models by local computations," *Computational Statisticals Quarterly*, vol. 4, 1990, pp. 269–282.

[19] P. Judea, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Elsevier Science, 1988.

[20] M. Koivisto, "Advances in exact Bayesian structure discovery in Bayesian networks," *In Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2006.

[21] M. Koivisto and K. Sood, "Exact Bayesian Structure Discovery in Bayesian Networks," *Journal of Machine Learning*, vol. 5, 2004, pp. 549–573.

[22] H. Liu, F. Hussain, C. L. Tan, and M. Dash, "Discretization: An Enabling Technique," *Data Mining and Knowledge Discovery*, vol. 6, 2002, p. 393C423.

[23] A. Moore and M. S. Lee, "Cached Suff cient Statistics for Eff cient Machine Learning with Large Datasets," *Journal of Artificial Intelligence Research*, vol. 8, 1998, pp. 67–91.

[24] A. Moore and W.-K. Wong, "Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning," *In International Conference. on Machine Learning*, 2003.

[25] R. E. Neapolitan, *Learning Bayesian Networks*, Prentice Hall, 2003.

[26] G. Rebane and J. Peal, "The Recovery of Casual Polytrees from Statistical Data," *In Proceedings Uncertainty in Artificial Intelligence*, 1987.

[27] R. Scheines, P. Spirtes, C. Glymour, and C. Meek, *Tetrad II: User Manual*, Hillsdale, New Jersery, Hillsdale, New Jersery, 1994.

[28] R. Shachter, "Probabilistic inference and inf uence diagrams," *Operations Research*, vol. 36, 1988, pp. 589–604.

[29] R. Shachter and C. Kenley, "Gaussian infuence diagrams," *Management Science*, vol. 35, 1989, pp. 527–550.

[30] T. Silander and P. Myllmaki, "A simple approach for f nding the globally optimal Bayesian network structure," *In Proceedings of Uncertainty in Artificial Intelligence*, 2006.

[31] A. Singh and A.W.Moore, *finding optimal bayesian networks by dynamic programming*, Tech. Rep., Carnegie Mellon University, 2005.

[32] P. Spirtes, C. Glymour, and R. Scheines, *Causation, Prediction and Search (2nd edition)*, MIT Press, 2001.

[33] J. Suzuki, "Learning Bayesian belief networks based on the MDL principle: An eff cient algorithm using the branch and bound technique," *In Proceedings of the Thirteenth International Conference on Machine Learning*, 1996.

[34] J. Tian, "A branch-and-bound Algorithm for MDL Learning Bayesian Networks," *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, 2000.

[35] I. Tsamardinos, L. Brown, and C. Aliferis, "The maxmin hill-climbing Bayesian network structure learning algorithm," *Machine learning*, 2006.

[36] T. Verma and J. Pearl, "Casual Networks: Semantics and Expressiveness," *In Proceedings Uncertainty in Artificial Intelligence*, 1988.

[37] N. Wermuth and S. L. Lauritzen, "Graphical and Recursive Models for Contingency Tables," *Biometrika*, vol. 72, 1983, pp. 537–552.