Ordering Estimation for Bayesian Network Structure Learning

Saulius Pačekajus
Aalborg University, Denmark

January 29, 2009

Abstract

Learning structure of Bayesian network have been a great challenge of Machine learning for the last few decades. A lot of ideas have been offered during that time and some of them proved to provide pretty handy results. The new idea of ordering estimation is proposed aiming to improve properties of ordering based or dependant Bayesian network structure learning algorithms. Three different approaches of ordering estimation are presented and tested by thorough experiments. The results show, that ordering based or dependant search algorithms can benefit from ordering estimation and even the idea of using K2 without optimal ordering is proposed.

Contents

Acknowledgments

I would like to thank my supervisor Yifeng and my girlfriend Agnė for the patience and efforts concerning this work.

Introduction

Bayesian Network (BN) is a probabilistic graphical model. It represents a joint probability distribution over a set of variables in a subject of matter. BNs are widely used in decision support systems, expert systems and adaptive systems because of their advantages – they can provide a solution under uncertainty, their structure can provide insights about the relations between variables and they can adapt to the changes of the subject of matter.

Learning structure of a Bayesian Network is a great challenge of Machine Learning. Choosing the best structure from all possible BN structures given the data is in fact proved to be an NP-complete task. Nevertheless there are a few heuristics invented, that aim to find optimal or close to optimal structure while using reasonable amounts of resources (data set size, time and computation power). Some of them are based or depends on the ordering of attributes of the domain.

This article presents three methods for ordering estimation (OE) that can improve results and usability of ordering based or dependant BN structure learning algorithms. The OE methods improve the quality of the resulting structure while being fast and simple. Extensive experiments (including sources) are provided to prove the utility of the methods as well as support the choice of particular estimation algorithms.

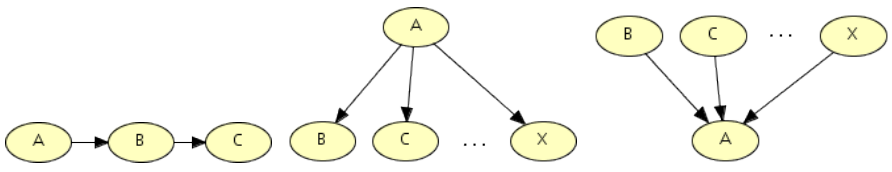
Chapter 1

Bayesian Networks

Formally, Bayesian belief network [11] is a directed acyclic graph (DAG) where nodes represent attributes or variables of a subject of matter and arcs between the nodes embody the causal relationship of attributes or variables. Every node of a network can gain one of the predefined values (states), depending on what kind of attribute it is representing – boolean, nominal or numeric. It is then said that an evidence is entered on a variable.

Node A is called a parent of node B if there exists an arc from node A to node B. The relation between two nodes is causal – specifically an arc from A to B is interpreted a conditional probability $P(A|B)$. There is also a conditional probability table (CPT) attached to every node of a Bayesian network. A CPT contains probability values for every state of a node for every possible parent value configuration. To conclude, a Bayesian network represents a joint distribution over the attributes of a subject of matter, so $P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|Pa(X_i))$, where $\mathbf{X} = \{X_1, \cdots, X_n\}$ are the variables of a network and $Pa(X_i)$ is a set of parents of variable X_i. An example of a Bayesian network, including CPTs for every node, can be seen in Figure 1.2.

Conditional independence properties of variables are represented by a graphical property of Bayesian network, called *d-separation*. There are three basic variable connection structures (Figure 1.1):



(a) Serial Connection (b) Diverging Connection (c) Converging Connection

Figure 1.1: Connection types

- *Serial connection*: is a type of connection presented in Figure 1.1a. If the state of B is unknown, a knowledge about A will influence a knowledge about C and in the other way around – a knowledge about C will affect a knowledge about A. Nevertheless if the state of B is known, then the

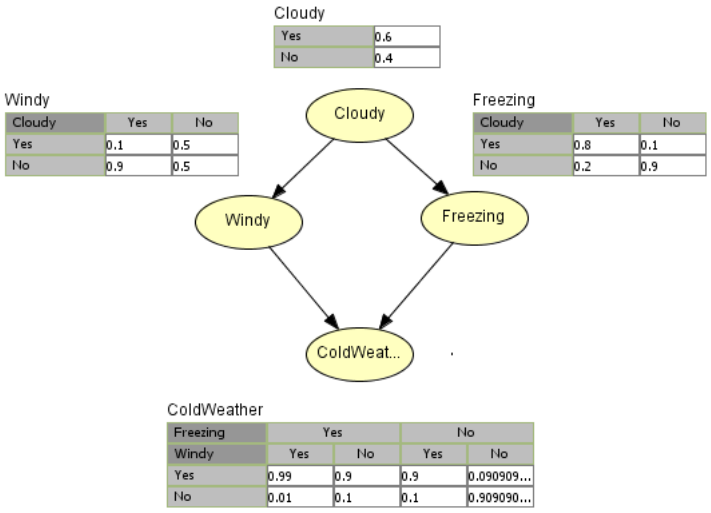


Figure 1.2: A small Bayesian belief network with conditional probability tables for every node.

channel between A and C is blocked and no evidence on A will affect C and in the other way around. A and C are called *d-separated given* B.

- *Diverging connection*: Figure 1.1b shows a type of connection that is called diverging. Just like in serial connection, influence can flow from any child of A to any other child as long as there is no evidence on A. $B, C, \cdots, X$ are called *d-separated given* A.

- *Converging connection*: a type of connection presented in Figure 1.1c has a bit different properties as the first two. Knowledge about any of the parents of A cannot affect knowledge about any other parent, unless there is an evidence on A or its descendants. $B, C, \cdots, X$ are called *d-separated* (and they are *d-connection given* A).

A more formal definition of Bayesian network and its properties can be found at [11].

Chapter 2

Structure Learning

Learning Bayesian network is a very important part of Machine learning. A formal definition of this process could be:

> Given a set of data, infer the topology for the causal network that may have generated them together with the corresponding uncertainty distribution. [18]

As stated in the definition, two stages of learning process are distinguished: *learning topology (structure)* and *learning parameters* of a network. While learning parameters is an important part of the process, the first stage – learning structure of BN – is much more complex task, hence it is the target of this work as well as most material in the area of BN learning. One interested in parameter learning should refer to [14, 19].

There are two main approaches for learning Bayesian network structure: *constraint based* approach and *score based* approach. The first one constructs the structure of the network based on the conditional tests between variables. The most famous constraint based algorithms are PC [20] and NPC [21]. The score based approach traverse DAG space led by data goodness-of-fit criterion [17]. The criterion is described by a scoring function that measures quality of the network structure given the data. Some representatives of this approach are structure learning algorithms K2 [8], B [3], FG [9] and HCMC [5].

Although both approaches are known to provide pretty good results, they also have well known drawbacks. Constraint based methods tend to need large data sets, the reasons for this, as stated in [17], are two: firstly to get reliable estimates from conditional tests between variables with weak conditional dependency; secondly when there is a high number of variables involved. Score based methods explore the space of all possible DAGs that grows more than exponentially to the number of variables, therefore score based methods use heuristics to explore only the part of the DAG-space that is likely to contain a BN structure, that is optimal given the data. The most widely used heuristic is a *hill-climber* (HC) greedy search that uses traversal operators to obtain network neighborhood [17] and choose one or more best candidates for further search.

Since the constraint based approach requires much larger dataset compared to the score based the latter have become the approach of choice in most of the works in the area of BN structure learning.

2.1 Score Based Learning

Generally a score based approach searches a DAG space (a set of all possible DAGs over some set of attributes) using the goodness-of-fit measure which simply is a score of the BN structure given the data. The higher is the score – the better is the structure.

Anyway the score alone is not enough. Generally it is computationally impossible to check all the possible DAG structures, because it is simply too much of them. The number of different DAGs given the number of nodes n is expressed by Robinson's formula [16] and is stated in equation 2.1.

$$G(n) = \begin{cases} 1 & \text{if } n = 0 \\ \sum_{i=1}^{n} (-1)^{i+1} \begin{pmatrix} n \\ i \end{pmatrix} 2^{i(n-1)} G(n-1) & \text{if } n > 0 \end{cases} \tag{2.1}$$

Since $G(n)$ is more than exponential to n, it is necessary to use some strategies to find the candidates that fit data best without checking all DAG space. A greedy search is a most popular solution in this situation. Generally it uses heuristics to traverse the search space by picking path of the best candidates at every step. Some usual search spaces are:

- The space of all possible DAGs given the attributes. It is usually traversed using arc operators like addition, deletion and reversion. See subsections 2.3.1 and 2.3.2 for more details and examples.
- The space of all possible attribute orderings. It is used in Ordering Search (OS) introduced in [22]. In OS it is traversed using simple swap operator which swaps two adjacent attributes in ordering. See subsection 2.3.3 for more details and examples.
- The space of equivalence classes. Equivalence class is a group of BNs that represent the same conditional independence relations. All the BNs from the same equivalence class are covered by the same partially directed graph. It can be traversed using the same arc operators like addition, deletion and reversion. See [6] and [15] for more information about equivalence classes and equivalence class based BN structure learning.

The quality measure often expressed as scoring function is the core of score based learning. Its main goal is to evaluate how good some BN structure represents conditional independence properties of the attributes of some given dataset. The following subsections introduce three quality measures used in this work. For simplicity, two usual assumptions should be taken into account for the rest of the article: all the attributes are discrete and there are no missing values in the data.

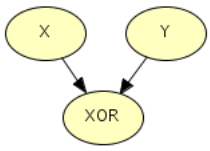
2.1.1 Mutual Information

Mutual information (MI) is a simple and natural way to evaluate conditional dependency of two variables. The higher MI is, the stronger dependence is between two variables.

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{2.2}$$

In general it is of the form of equation 2.2 where $P(x)$ is a probability of variable X having value x. However, as $P(x)$ is usually unknown when learning structure from the data, the observed frequency $\hat{P}(x)$ (number of entries where $X = x$ divided by total number of entries) is used.

Although MI can be used to evaluate the relationship between two variables, it is not sufficient to be used as the main quality measure for score based BN structure learning. The reason is that MI is a pairwise function while some parts of BN can embody more complex relations. Consider a logical XOR relation in Figure 2.1. The correct BN structure (Figure 2.1a) could not be learned using MI, because variables X, Y and XOR must be considered all together, otherwise $I(X; XOR) = I(Y; XOR) = 0$ would indicate, that X and XOR as well as Y and XOR are conditionally independent, thus no arc would be added between X and XOR as well as Y and XOR. Therefore MI is not usually concerned as a scoring function. Nevertheless, MI can be successfully used while approximating initial parent candidates. An example of that could be a Sparse Candidate algorithm (restriction step) introduced in [10].



(a) BN representation of XOR relation

X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

(b) CPT of XOR node

Figure 2.1: XOR relation

In order to overcome limits of MI measure, more advanced metrics have been introduced. The most popular two of them are described in the following subsections.

2.1.2 Bayesian Inference Measure

In Bayesian Inference the score is a probability of the network structure given the data ($P(B_S|D)$). According to a basic probability rule $P(B_S|D) = \frac{P(B_S,D)}{P(D)}$, therefore in order to maximize $P(B_S|D)$ it is enough to maximize $P(B_S, D)$.

$$P(B_S, D) = P(B_S) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{2.3}$$

Bayesian Inference uses Bayesian Dirichlet scoring function (BDe) introduced in [8]. It has a form of equation 2.3 for discrete variables, where B_S is a structure of Bayesian network, D is a data set, $P(B_S)$ is the prior probability of a network structure B_S, that can be used to express any prior knowledge of a structure. n is the number of variables, r_i is the arity of a variable X_i, q_i is the number of possible parent ($Pa(X_i)$) configurations, N_{ijk} is a number of entries in D where variable X_i is set to its k^{th} value and its parents are instantiated to their j^{th} configuration and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

Since the equation involves factorials and runs through all of the parent configurations it is obviously a time costly operation and the results (depending

on the number of attributes and entries in data set) are usually extremely small, therefore a logarithm of the score is often used instead. It replaces multiplication with summation and provides numbers that are easier to process. Assuming there is no prior knowledge about the structure being considered, $P(B_S)$ can be set to 1. Therefore the implementation version of the score function can be expressed in equation 2.4.

$$\log P(B_S, D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \left[\log((r_i - 1)!) - \log((N_{ij} + r_i - 1)!) + \sum_{k=1}^{r_i} \log(N_{ijk}!) \right] \tag{2.4}$$

It can also be seen, that both – the original BDe and its logarithmic version used in implementation are decomposable [17]. It means, that the score of overall (global) structure is a product or sum of local structure (child-parent configurations) scores. This property is widely used to improve the efficiency of a greedy search – it is enough to calculate the local score difference to know the score difference of all the network.

A more detailed discussion on Bayesian Inference measure and BDe can be found in [8].

2.1.3 Minimal Description Length

The Minimal Description Length principle employs some ideas from coding theory – it searches for a BN structure that has the shortest sum of lengths of the encodings of a BN and the data given a BN. So generally the MDL measure is the number of bits needed to encode a BN and a data set given a BN.

$$L(B_S, D) = \log P(B_S) - N \cdot H(B_S, D) - \frac{1}{2} K \cdot \log N \tag{2.5}$$

Bouckaert in [2] provides a definition of MDL measure as in equation 2.5, where $H(B_S, D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} -\frac{N_{ijk}}{N} \log \frac{N_{ijk}}{N_{ij}}$ and $K = \sum_{i=1}^{n} q_i(r_i - 1)$ and $B_S, P(B_S), D, n, r_i, q_i, N_{ijk}, N_{ij}$ the same as in BDe definition above.

The first term of the equation 2.5 takes the same role as the $P(B_S)$ part in BDe – it embodies any prior knowledge about the structure B_S. The second term $N \cdot H(B_S, D)$ represents the conditional entropy of the structure B_S. Finally, the last term $\frac{1}{2} K \cdot \log N$ is the error, which is introduced by estimating all the required probabilities B_P for the structure B_S from the data set D in $H(B_S, D)$, where K is the number of independent probabilities that have to be estimated.

As it can be seen from the equation 2.5, the tradeoff between accuracy and complexity of a structure B_S is introduced in MDL measure. Entropy $H(B_S, D)$ is smaller when the number of arcs is bigger, because the structure with more arcs represents the probability distribution in D more accurately. On the other hand K is bigger when the number of arcs is bigger, because every new arc increases complexity of the structure. So in order to maximize the measure $L(B_S, D)$ it is necessary to minimize both – $H(B_S, D)$ and K, therefore every new arc must decrease the entropy $H(B_S, D)$ enough to overcome the increase of K.

$$L(B_S, D) = \log P(B_S) + \sum_{i=1}^{n} \left(\sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}} - \frac{1}{2} q_i (r_i - 1) \right) \quad (2.6)$$

After expressing MDL measure as in equation 2.6 it can be clearly seen, that $L(B_S, D)$ is also decomposable, which means that the local score changes are equal to the changes of global score.

Since both measures – MDL and BDe – look similar, one might wonder if they actually are equal and at some level one would be right. Bouckaert in [2] has shown, that the MDL measure ($L(B_S, D)$) is actually the approximation of the logarithm of BDe ($\log P(B_S, D)$). Therefore they can both be used in more or less the same methods.

One interested to learn more about MDL should refer to [13], [2] or [17].

2.2 Ordering in Score Based Learning

Another quite important concept in score based learning is variable ordering. It is usually used to constraint a search space by a simple rule – variable X can only consider another variable Y to be its parent ($Y \in Pa(X)$) if and only if Y precedes X given ordering.

For example, consider ordering of four variables (A, B, C, D). Then $Pa(A) =$, $Pa(B) \in \mathcal{P}(\{A\})$, $Pa(C) \in \mathcal{P}(\{A, B\})$ and $Pa(D) \in \mathcal{P}(\{A, B, C\})$, where $\mathcal{P}$ denotes power set.

The benefits of having variable ordering in the process of BN structure learning as noted in [22] are that it reduces search space and helps to avoid expensive acyclicity checks.

2.3 Algorithms

Some BN structure learning algorithms are introduced in this subsection. While K2 (2.3.1) and hill climber (2.3.2) are two classical examples of score based BN structure learning, ordering based search (2.3.3) aims to solve the same problem in different manner.

2.3.1 K2

K2 is one of the first score based BN structure score algorithms introduced in [8]. It needs optimal variable ordering and a score function to calculate local structure. Although originally it uses Bayesian Inference measure in [8], MDL can also be used in the same manner.

As it can be seen from K2 pseudo code in Figure 2.2, the algorithm iterates through all n variables and selects at most k parents from at most $n - 1$ candidates. Hence the overall complexity of the algorithm is $O(kn^2)$ which makes it the fastest score based BN structure learning algorithm (see subsections 2.3.2 and 2.3.3 for comparison). Its complexity is comparatively small because it uses an optimal ordering of variables to constraint the DAG space that is being searched – the algorithm avoids an expensive step selecting valid parameters

Require: $\mathbf{X} = \{X_1, \cdots, X_n\}$ – a set of variables, $\prec$ – an optimal ordering of variables in $\mathbf{X}$, D – a data set, g – a decomposable scoring function, k – a maximum number of parents and a function $Prec(X_i) = \{X : X \prec X_i\}$
Ensure: for each variable X_i an optimal parent set $Pa(X_i)$ ($|Pa(X_i)| \leq k$)

```
for X_i in X do
    Pa(X_i) = ∅
    S_current = g(X_i, Pa(X_i), D)
    proceed = true
    while proceed and |Pa(X_i)| ≤ k do
        X_z = argmax_{X ∈ Prec(X_i) \ Pa(X_i)} g(X_i, Pa(X_i) ∪ X, D)
        S_new = g(X_i, Pa(X_i) ∪ X_z, D)
        if S_new > S_current then
            Pa(X_i) = Pa(X_i) ∪ X_z
            S_current = S_new
        else
            proceed = false
        end if
    end while
end for
```

Figure 2.2: The pseudo code of K2 algorithm

for traversal operators (see subsection 2.3.2 for more details). Unfortunately optimal ordering is usually not available thus making K2 impractical in most situations.

2.3.2 Hill Climber Greedy Search

Hill climber greedy search (HC) is a general heuristic for score based DAG search. In other words it is a class of many similar algorithm, starting with very simple B [3], FG [9] and including some advanced and complicated HCMC [5] and ORSearch [1]. While varying in difficulty and resulting structure quality, most of them can be represented by a simplified generic HC algorithm described in Figure 2.3.

A new element *scenario* is introduced for convenience – it can be viewed as a simple structure containing three fields: *operator* – an operator being applied, *parameters* – parameters for operator being applied and *gain* – a score gain of the operator being applied.

While it might look quite sophisticated from the pseudo code in Figure 2.3, it is much easier to imagine when having particular parameters in mind. Consider OP contains the three usual arc operators – addition, deletion and reversal. Then a typical scenario would be

$$\begin{bmatrix} operator: & Addition \\ parameters: & \{X_i, Xj\} \\ gain: & g(X_i, Pa(X_i) \cup X_j, D) - g(X_i, Pa(X_i), D) \end{bmatrix}$$

The function $GetS(op)$ depending on op would return: scenarios for every two nodes where arc can be added without introducing a cycle, if $op = Addition$;

Require: $\mathbf{X} = \{X_1, \cdots, X_n\}$ – a set of variables, D – a data set, g – a decomposable scoring function, OP – a set of traversal operators, $GetS(op)$ – a function returning all valid scenarios for operator op being applied on DAG

Ensure: DAG – a DAG representing locally optimal structure over $\mathbf{X}$ given data D

1: DAG – an arcless DAG over $\mathbf{X}$
2: $S_{current} = g(DAG, D)$
3: $proceed =$ **true**
4: **while** $proceed$ **do**
5: $SC = \emptyset$
6: **for** $op \in OP$ **do**
7: $SC_{op} = GetS(op)$
8: **for** $s \in SC_{op}$ **do**
9: let L_{before} be a local structure targeted by s before applying op
10: let L_{after} be a local structure targeted by s after applying op
11: $s.gain = g(L_{after}, D) - g(L_{before}, D)$
12: **end for**
13: $SC = SC \cup SC_{op}$
14: **end for**
15: $s_{best} = \text{argmax}_{s \in \mathbf{SC}} s.gain$
16: **if** $s_{best}.gain > 0$ **then**
17: apply $s_{best}.operator$ on DAG given $s_{best}.parameters$
18: $S_{current} = S_{current} + s.gain$
19: **else**
20: $proceed =$ **false**
21: **end if**
22: **end while**

Figure 2.3: The pseudo code of generic HC algorithm

scenarios for every arc, if $op = Deletion$; scenarios for every arc where it can be reversed without introducing a cycle, if $op = Reversal$.

Having the same usual three operators in mind, the complexity of the function $GetS(op)$ is $O(n^2)$ as there are at most $n(n-1)$ arcs to be evaluate for every operator. Thus every iteration of the outer loop is $O(3n^2) = O(n^2)$ complex which is more or less equivalent to running K2 with maximum number of parents $k = 3$, which is quite realistic. Nevertheless HC usually iterates more than once (unless it starts with optimal DAG), thus it tends to need more time than K2.

For more profound explanation of hill climber greedy search in general and particular algorithms and their implementation details one should refer to [17], [3], [9], [5] and [1].

2.3.3 Ordering Based Search

Ordering based search (OS) addresses the problem of learning optimal BN structure given the data in a different way compared with K2 and HC. Instead of traversing DAG search space, it traverses the space of attribute orderings. According to [22] it does so, because ordering search space is much smaller than

DAG search space, the search steps are more global (hence it is supposed to escape local optima better) and it avoids expensive acyclicity checks.

Require: all the same as in main algorithm in Figure 2.6, X – a variables, C – a set of parent candidates and k – a maximum number of parents
Ensure: *best* – best parent set for variable X given candidates C
1: $\mathbf{P} = \{P : P \in \mathcal{P}(C), |P| \leq k\}$
2: $best = \text{argmax}_{P \in \mathbf{P}} g(X, P, D)$
3: **return** *best*

Figure 2.4: The pseudo code of *best* function used in OS algorithm

Require: all the same as in main algorithm in Figure 2.6, X_i, X_j – two adjacent variables ($j = i + 1$), $Pa(X_i), Pa(X_j)$ already computed and $\prec$ – ordering of variables
Ensure: *gain* – a score gain of swapping X_i and X_j in ordering $\prec$
1: $Pa'(X_i) = best(X_i, Prec(X_i, \prec) \cup X_j, k)$
2: $Pa'(X_j) = best(X_j, Prec(X_j, \prec) \backslash X_i, k)$
3: $gain_i = g(X_i, Pa'(X_i), D) - g(X_i, Pa(X_i), D)$
4: $gain_j = g(X_j, Pa'(X_j), D) - g(X_j, Pa(X_j), D)$
5: $gain = gain_i + gain_j$
6: **return** *gain*

Figure 2.5: The pseudo code of *gain* function used in OS algorithm

As OS is still a score based search it also uses score which is the score of the best structure consistent with given ordering. The method for doing that as proposed in [22] is simply to select the best possible parent set given ordering for every node. The function *best* in Figure 2.4 does exactly that, where $\mathcal{P}(C)$ means the power set of C. Obviously the number of parent sets to evaluate for variable X_i is $\sum_{l=0}^{k} \binom{i}{l}$, where k is the maximum number of parents. As $\binom{n}{k} = O(n^k)$, the complexity of the overall function *best* is also $O(n^k)$.

Pseudo code of the OS algorithm can be seen in Figure 2.6. The algorithm first precomputes best parent sets for every variable and evaluates the gain of swapping two adjacent variable in the ordering. It then loops as long as the best gain is positive. The algorithm applies the best gain swap on the current ordering, updates the parent sets of swapped variables and evaluates two new swap possibilities at every iteration.

In order to make it clear how the algorithm works, consider a small example of the BN over four variables A, B, C, D (see Figure 2.7a). Assume the original BN structure in Figure 2.7a is the optimal structure, hence it is a desired result of OS. Then there are 4 optimal orderings for it: (A, C, B, D), (A, C, D, B), (C, A, B, D), (C, A, D, B). Ideally it should be, that $gain(A, C) = gain(C, A) = gain(B, D) = gain(D, B) = 0$, $gain(B, A)$, $gain(D, A)$, $gain(B, C)$, $gain(D, C) > 0$ and $gain(A, B)$, $gain(A, D)$, $gain(C, B)$, $gain(C, D) < 0$. Consider non-optimal random orderings (D, B, C, A) and (A, D, B, C) – OS algorithm should take learning paths depicted in Figures 2.7b and 2.7c respectively.

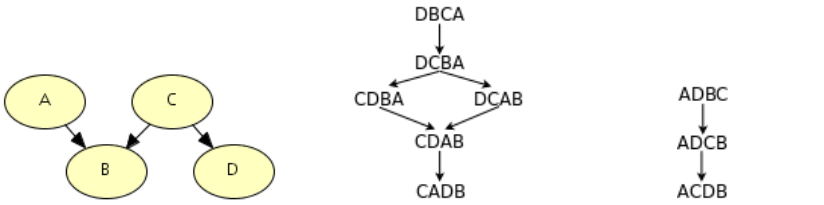
Require: $\mathbf{X} = \{X_1, \cdots, X_n\}$ – a set of variables, D – a data set, g – a decomposable scoring function, k – a maximum number of parents, a function $Prec(X_i, \prec) = \{X : X \prec X_i\}$, $gain(X_i, X_j, \prec)$ – a function returning score gain of swapping X_i and X_j in $\prec$ (see Figure 2.5), $best(X, C)$ – a function selecting best X parent set given candidates C (see Figure 2.4)

Ensure: DAG – a DAG representing optimal structure over $\mathbf{X}$ given data D

let $\prec$ be some random ordering of variables in $\mathbf{X}$
for $X_i \in \mathbf{X}$ **do**
  $Pa(X_i) = best(X_i, Prec(X_i, \prec), k)$
  if $i < n$ **then**
    $gain_{i(i+1)} = gain(i, i+1, \prec)$
  end if
end for
$proceed =$ **true**
while *proceed* **do**
  $i, j = \text{argmax}_{i,j} gain_{ij}$
  if $gain_{ij} > 0$ **then**
    swap X_i and X_j in $\prec$
    swap i and j
    $Pa(X_i) = best(X_i, Prec(X_i, \prec), k)$
    $Pa(X_j) = best(X_j, Prec(X_j, \prec), k)$
    $gain_{ij} = -gain_{ji}$
    $gain_{(i-1)i} = gain(i-1, i, \prec)$
    $gain_{j(j+1)} = gain(j, j+1, \prec)$
  else
    $proceed =$ **false**
  end if
end while
construct DAG according to $Pa(X_i)$ for every $X_i \in \mathbf{X}$

Figure 2.6: The pseudo code of OS algorithm

While choosing best parent sets and evaluating swap possibilities are separated for the sake of clarity in Figure 2.6, both of them can be optimized by merging in order to have only one run over possible parent sets. Even though it does not decrease the complexity of the algorithm which is $O(n^{k+1})$ for initialization and $O(n^k)$ for every iteration. Although the author in [22] claims that OS can be faster and produce higher quality structures than HC, logic however is against that claim – OS is a more complex algorithm than HC, given $k > 2$ and this fact is supported by experiments in section 4.



(a) A Bayesian network

(b) Ordering learn path for random ordering (D, B, C, A)

(c) Ordering learn path for random ordering (A, D, B, C)

Figure 2.7: OS example

Chapter 3

Ordering Estimation

The aim of Ordering Estimation (OE) method is to improve ordering based or dependant BN structure learning algorithms like K2 and OS. While optimal ordering is one of the requirements for K2, OS can also benefit from that, because optimal ordering could simply shorten the search path (as it is already what OS searches for) by reducing the number of iterations needed. The quality of the resulting BN structure is also expected to increase since again, in K2 optimal ordering is one of the requirements to learn optimal BN structure and in the case of OS it should be easier to find higher quality ordering and escape local minimas as the search already starts with higher quality ordering and it does not take steps that could decrease it.

To find an optimal ordering by exhaustive search is in fact computationally impossible as the number of all possible orderings over n variables is $O(n^n) = 2^{O(n \log_2 n)}$. Again, just like in the case of optimal BN structure search – heuristics could be used and that is actually what OS do. Nevertheless, the complexity of OS ($O(n^{k+1})$ for initialization and $O(n^k)$ for every iteration) is already impractical. Therefore the purpose of OE is not to find an optimal ordering, but to estimate a *good ordering* as close to optimal as it is possible by using reasonable computational resources. A *good ordering* can be defined as the ordering which can be used in ordering based or dependant structure learning algorithms to get higher average score compared with random ordering without loosing much time.

In order to estimate ordering it is necessary to have some criterion based on which variables are ordered. Consider a weight set $W = \{w_1, \cdots, w_n\}, \forall i \in \{1, \cdots, n\}, w_i \in \mathbb{R}$ where w_i represents the weight of variable X_i. These weights can be used to define variable ordering in the way, that $X_i \in \mathbf{X}$ precedes $X_j \in \mathbf{X}$ if and only if $w_i > w_j$. Hence variable X_i can only be a parent of X_j if $w_i > w_j$. Therefore estimating optimal ordering can be restated as estimating weights W such that for every $X_j \in \mathbf{X}$ and for every $X_i \in \mathbf{Pa}(X_j)$ holds $w_i > w_j$.

It is hard to imagine how can this be done without building BN structure itself, but remember that the goal is to estimate a *good ordering* (not necessarily optimal). Therefore the accuracy of estimating W can be relaxed as long as the resulting ordering is *good ordering*.

Intuitively, weights W must contain values for every node starting with the highest values for top parents and finishing with the lowest values for bottom children. Three methods aiming to do that are introduced in the following

sections.

3.1 OE1 – Scoring Function Approach

The first ordering estimation method (OE1) aims to build weights W using scoring function (see subsection 2.1). It simply counts the positive, zero and negative score gains of every variable X_i being a single parent of every other variable $X_j \in \mathbf{X} \backslash X_i$ (equation 3.1).

$$w_i = \sum_{j}^{\{1,\cdots,n\}\backslash i} sgn(g(X_j, \{X_i\}, D) - g(X_j, \emptyset, D)) \tag{3.1}$$

OE1 comes with the naive assumption that a parent or a grandparent has better chances to be a parent than a child or a grandchild. This way, the closer to the top variable is, the more other variables find it good as a parent (positive and higher score gain) and the closer to the bottom variable is, the more other variables find it bad as a parent (negative or low score gain). The *sgn* of a score gain is taken, because local scores of different structures cannot be compared.

The overall algorithm can be seen in Figure 3.1.

Require: $\mathbf{X} = \{X_1, \cdots, X_n\}$ – a set of variables, D – a data set, g – a decomposable scoring function
Ensure: $\prec$ – estimated ordering
1: $W = \{w_1, \cdots, w_n\}$ a set of weights, where w_i is the weight of X_i
2: **for** $w_i \in W$ **do**
3: $\quad w_i = 0$
4: **end for**
5: **for** $X_i \in \mathbf{X}$ **do**
6: $\quad$ **for** $X_j \in \mathbf{X} \backslash \{X_i\}$ **do**
7: $\quad\quad w_j = w_j + sgn(g(X_i, \{X_j\}, D) - g(X_i, \emptyset), D))$
8: $\quad$ **end for**
9: **end for**
10: $\prec$ – a new ordering for variables in $\mathbf{X}$
11: sort $\prec$ according to weights in W

Figure 3.1: The pseudo code of OE1 algorithm

3.2 OE2 – Mutual Information Approach

The second ordering estimation method (OE2) is similar to OE1. The only difference is that it sums mutual information (MI) of every variable X_i pairwise with every other variable $X_j \in \mathbf{X} \backslash X_i$ (equation 3.2).

$$w_i = \sum_{j}^{\{1,\cdots,n\}\backslash i} MI(X_i, X_j, D)) \tag{3.2}$$

Again OE2 comes with an assumption that a parent or a grandparent has better chances to get higher MI values with other variables than a child or a grandchild.

The overall algorithm can be seen in Figure 3.2.

Require: $\mathbf{X} = \{X_1, \cdots, X_n\}$ – a set of variables, D – a data set, m – a mutual information function (see equation 2.2 in subsection 2.1.1).
Ensure: $\prec$ – estimated ordering
1: $W = \{w_1, \cdots, w_n\}$ a set of weights, where w_i is the weight of X_i
2: **for** $w_i \in W$ **do**
3: $w_i = 0$
4: **end for**
5: **for** $X_i \in \mathbf{X}$ **do**
6: **for** $X_j \in \mathbf{X} \backslash \{X_i\}$ **do**
7: $w_j = w_j + m(X_i, X_j, D)$
8: **end for**
9: **end for**
10: $\prec$ – a new ordering for variables in $\mathbf{X}$
11: sort $\prec$ according to weights in W

Figure 3.2: The pseudo code of OE2 algorithm

3.3 OE3 – Dependence Tree Approach

The third ordering based method (OE3) is a bit more advanced than OE1 and OE2. It uses Dependence Tree (DT) to get weights W. DT is a graphical model introduced by [7], where the author shows that a discrete n-dimensional probability distribution can be approximated by product of second-order distributions in a form of equation 3.3 where $P(X_i|X_0) = P(X_i)$.

$$P((X)) = \prod_{i}^{n} P(X_i|X_{j(i)}), 0 \leq j(i) < i \tag{3.3}$$

In fact a pair consisting of the variable set $(X) = \{X_1, \cdots, X_n\}$ and the mapping $j(i)$ where $0 \leq j(i) < i$ is called the dependence tree of the distribution P. It can be represented as a spanning tree in a directed graph where nodes $X_1, \cdots, X_n$ are represented by vertices and there exists a directed arc from X_i to X_k for every pair of variables X_i and X_k such that $k = j(i)$. An example of dependence tree for probability distribution $P(\mathbf{X}) = P(X_2|X_1)P(X_3|X_2)P(X_4|X_2)P(X_5|X_2)P(X_6|X_5)$ can be found in Figure 3.3.

While there are n^{n-2} possible dependence trees for n variables, the optimal one, that approximates probability distribution best is desired. Mutual information (MI) of every pair of variables that have arc between them is used as a weight of the arc. As stated in [7] the maximum weight dependence tree is an optimal approximation of $P(\mathbf{X})$. Now the problem of finding a maximum weight dependence tree can be seen as very similar to finding minimal spanning tree in a directed graph. The only difference is that it should pick arcs with maximum instead of minimum weights. As proposed in [7], Kruskal's algorithm ([12]) which is $O(n^2 \log n)$ complex is used for that.

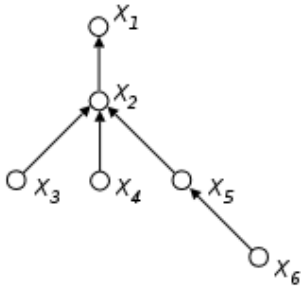


Figure 3.3: Example of a dependence tree

The OE3 method comes with the assumption, that the closer to the top node is, the more general variable it represents and the closer to the bottom it is, the more specific variable it represents. This is assumed for both – BN as well as DT, but since DT is easier to construct, the goal of OE3 is to simply come up with an optimal ordering of DT. As DT is also DAG, the definition of optimal ordering is the same as for BN.

Algorithm for OE3 can be seen in Figure 3.4.

Require: $\mathbf{X} = \{X_1, \cdots, X_n\}$ – a set of variables, D – a data set, m – a mutual information function (see equation 2.2 in subsection 2.1.1).
Ensure: $\prec$ – estimated ordering

1: $W = \{w_{ij} : i, j \in \{1, \cdots, n\}, i < j \leq n\}$, where w_{ij} is the weight of arc $X_i \rightarrow X_j$
2: **for** $\forall i \in \{1, \cdots, n\}$ **do**
3: **for** $\forall j \in \{i+1, \cdots, n\}$ **do**
4: $w_{ij} = m(X_i, X_j, D)$
5: **end for**
6: **end for**
7: DAG – an arcless DAG over $\mathbf{X}$
8: $\mathbf{C} = \{C_i = \{X_i\} : i \in \{1, \cdots, n\}\}$, were C_i is set of variables X_i has a path to
9: **while** $W \neq \emptyset$ **do**
10: $i, j = argmax_{ij} w_{ij}$
11: **if** $X_j \notin C_i$ **then**
12: add $X_i \rightarrow X_j$ to DAG
13: $C_i = C_j = C_i \cup C_j$
14: $W = W \backslash \{ws_i\}$
15: **end if**
16: **end while**
17: $\prec = \emptyset$ – a new ordering
18: **while** $| \prec | < n$ **do**
19: **for** $X_i \in \mathbf{X}$ **do**
20: **if** $X_i \notin \prec$ and $\forall X_j \in \prec$: $X_j \in \mathbf{Pa}(X_i)$ **then**
21: add X_i to the rightmost of $\prec$
22: **end if**
23: **end for**
24: **end while**

Figure 3.4: The pseudo code of OE3 algorithm

Chapter 4

Experiments

Two kinds of experiments have been conducted: the first to check the assumption of OE1 and OE2 and the second to test how proposed OE algorithms work. Four Bayesian networks have been used to generate test cases: Studfarm (12 nodes), Boblo (22 nodes), Boerlage92 (22 nodes) and Alarm (37 nodes). Table 4.1 shows how many different data sets of different sizes have been used during experimenting.

Studfarm		Boblo		Boerlage92		Alarm	
Entries	Sets	Entries	Sets	Entries	Sets	Entries	Sets
2000	100	2000	100	2000	10	2000	10
4000	100	4000	100	4000	10	4000	10
8000	100	8000	100	8000	10	8000	10
16000	100	16000	100	16000	10	16000	10
		32000	100	32000	10	32000	10
		64000	100				

Table 4.1: Datasets used

4.1 Assumptions of OE1 and OE2

The goal of the first part of experiments is to check the assumption introduced in subsections 3.1 and 3.2. It states, that in BN the closer to the top a variable is, the more other variables find it good as a parent and the closer to the bottom it is, the less other variables find it good as a parent. According to this assumption, variables closer to the top are expected to get more positive and higher score gains while being evaluated as parents to other variables in case of OE1 and higher MI values while being evaluated with other variables in case of OE2. So the task for experiments is simply to compare the weights generated by OE1 and OE2 to some attributes of variables that describe their position between the top and the bottom of original BN. The numbers of ancestors and successors are expected to do that because obviously, variables closer to the top have less ancestors and might (although need not to) have more successors. Therefore experiment simply uses a data sets to generate OE1 and OE2 weights

for every variable, averages them and calculates the correlation to the numbers of ancestors and successors for every variable.

Variable	Successors	Ancestors	OE1 weight	OE2 weight
K	3	0	-5	0.0278
John	0	11	9	0.1036
Gwenn	2	2	2	0.0826
Irene	1	6	8	0.0955
Dorothy	2	2	4	0.0952
Henry	1	5	6	0.1010
Brian	5	0	-1	0.0561
Eric	2	2	0	0.0717
Ann	6	0	0	0.0843
Fred	2	2	2	0.0825
Cecily	3	0	-5	0.0278
L	3	0	-5	0.0284
Correlation with Successors			-0.64	-0.45
Correlation with Ancestors			0.87	0.69

Table 4.2: OE1 and OE2 assumption check results for Studfarm

As it can be seen in the results tables (Table 4.2 – 4.5), the correlation between OE1 weights and the number of ancestors of variable is pretty high for BNs Studfarm (Table 4.2), Boblo (Table 4.3) and Alarm (Table 4.5). Unfortunately, as it can be seen in Table 4.4, there is no correlation in case of BN Boerlage92. OE2 weights, on the other hand, seem to have reasonable correlation only in case of Studfarm and Alarm as well as the same correlation (just positive) as OE1 weights in case of Boerlage92. It is hard to make any firm conclusion out of this alone before results of the second part of experiments are present.

4.2 Ordering Estimation in Action

The second and more important part of experiments aims to test if ordering estimation methods really work. Additionally, reversed ordering for every OE method was also tested. Call them OE1R, OE2R and OE3R respectively.

During experiment phase, tests are run for all datasets. AD-trees are build for every dataset in order to boost up score function evaluation time as it efficiently caches all data frequency counts. Refer to [4] for more information concerning AD-trees. A single step can be described as:

1: Build AD-tree for a dataset
2: Execute OS with optimal ordering
3: Execute OS with ordering prepared by OE1
4: Execute OS with ordering prepared by OE1R
5: Execute OS with ordering prepared by OE2
6: Execute OS with ordering prepared by OE2R
7: Execute OS with ordering prepared by OE3
8: Execute OS with ordering prepared by OE3R

Variable	Successors	Ancestors	OE1 weight	OE2 weight
f3	0	8	8	1.3000
f2	0	8	7	0.1800
asts2	0	3	-6	1.2600
asts1	0	3	-6	1.2600
astd2	0	3	-6	1.3000
astd1	0	3	-6	1.3100
sc	2	1	-16	0.1000
dc	2	1	-16	0.0600
pe	6	0	-16	0.1300
f1	0	8	7	1.3700
atd2	7	0	-6	1.3100
atd1	7	0	-6	1.3100
ats2	7	0	-6	1.2800
ats1	7	0	-6	1.2800
aph2	5	2	-2	1.8300
aph1	5	2	-2	1.8400
ageno	4	6	7	3.2700
f4	0	8	7	1.0400
node20	1	0	-19	0.0000
node21	1	0	-18	0.0000
node22	1	0	-19	0.0000
node23	1	0	-19	0.0000
Correlation with Successors			-0.07	0.29
Correlation with Ancestors			0.86	0.36

Table 4.3: OE1 and OE2 assumption check results for Boblo

9: Execute OS 20 times (5 in case of Boerlage92 and Alarm) with random ordering
10: Execute K2 with optimal ordering
11: Execute K2 with ordering prepared by OE1
12: Execute K2 with ordering prepared by OE1R
13: Execute K2 with ordering prepared by OE2
14: Execute K2 with ordering prepared by OE2R
15: Execute K2 with ordering prepared by OE3
16: Execute K2 with ordering prepared by OE3R
17: Execute K2 20 times (5 in case of Boerlage92 and Alarm) with random ordering
18: Execute HC (see section 4.3)

As maximum number of parents is required for OS and K2, the actual numbers where used: 2 for Studfarm and Boblo and 4 for Boerlage92 and Alarm.

The results are average, variance and standard deviation of resulting BN score, run time and iteration count in case of OS. The spreadsheet containing complete results should be available on the project website at http://cs.aau.dk/~cerberus. Because of limitations of this print form, only the most important part of the results is present in this paper.

Variable	Successors	Ancestors	OE1 weight	OE2 weight
C21	0	20	-13	0.0600
C19	4	13	-6	0.6300
C18	5	12	-6	0.6300
C16	6	11	0	0.2200
C14	9	5	3	0.3400
C13	1	8	-10	0.0700
C12	10	3	-16	0.0300
C11	12	4	2	0.4100
C10	2	7	-1	0.2400
C8	14	3	-6	0.0900
C7	11	2	-6	0.1000
C6	21	0	-10	0.0900
C5	20	1	1	0.1500
C4	17	2	2	0.3800
C3	4	3	-6	0.3600
C2	0	5	-6	0.3500
C1	1	0	-19	0.0100
C22	9	6	-13	0.0500
C23	8	8	0	0.2800
C24	7	9	-4	0.1600
C25	3	14	-6	0.2300
C26	2	15	-10	0.1900
C27	1	16	-13	0.1000
Correlation with Successors			0.38	-0.06
Correlation with Ancestors			-0.15	0.15

Table 4.4: OE1 and OE2 assumption check results for Boerlage92

MinVol	0	7	3	3.4600
Press	0	6	5	2.9100
PAP	0	1	-31	0.0400
ExpCO2	0	9	5	4.0300
HRBP	0	20	14	3.1200
ErrLowOutput	1	0	-33	0.0300
HRSat	0	20	14	2.8800
HREKG	0	20	13	2.8800
ErrCauter	2	0	-31	0.0700
BP	0	23	17	1.0600
History	0	1	-22	0.3100
CO	1	22	24	2.9500
HR	5	18	13	3.5300
Catechol	6	17	15	3.0600
ArtCO2	8	8	5	3.6500
TPR	7	1	-19	0.6500
Anaphylaxis	8	0	-24	0.0300
SaO2	7	12	10	3.5000
PVSat	8	9	8	3.8500
FiO2	9	0	-22	0.0400
VentAlv	11	7	5	4.6000
VentLung	13	6	3	3.9500
VentTube	15	3	0	2.4600
Disconnect	16	0	-4	0.9000
VentMach	16	1	-6	0.9900
MinVolSet	17	0	-9	0.4300
KinkedTube	15	0	-6	0.5100
Shunt	8	2	-4	0.5600
Intubation	16	0	-4	1.0400
PulmEmbolus	10	0	-21	0.0600
InsuffAnesth	7	0	-21	0.3800
PCWP	0	3	-19	1.7800
CVP	0	3	-20	1.6600
StrokeVolume	2	2	-20	1.4000
LVEDVolume	2	2	-20	2.0900
LVFailure	7	0	-22	0.3700
Hypovolemia	6	0	-23	1.5100
Correlation with Successors			-0.01	-0.15
Correlation with Ancestors			0.84	0.62

Table 4.5: OE1 and OE2 assumption check results for Alarm

	Optimal		OE1		OE1R		OE2		OE2R		OE3		OE3R		Random	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	-843	7	**-844**	6	-877	5	-855	6	-878	6	-870	7	-881	7	-876	7
4000	-1601	7	**-1602**	7	-1665	7	-1622	7	-1667	7	-1642	9	-1684	9	-1667	8
8000	-3111	10	**-3111**	9	-3226	9	-3134	10	-3238	10	-3190	12	-3272	12	-3233	12
16000	-6121	8	**-6121**	7	-6336	8	-6150	8	-6354	8	-6281	10	-6436	10	-6358	10

Table 4.6: Experiment results for network Studfarm running OS algorithm

	Optimal		OE1		OE1R		OE2		OE2R		OE3		OE3R		Random	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	-844	1	-856	1	-888	1	**-844**	1	**-844**	1	-899	1	-929	1	-909	1
4000	-1602	1	-1611	1	-1688	1	**-1602**	1	**-1602**	1	-1712	2	-1797	2	-1743	1
8000	-3111	2	-3113	2	-3275	2	**-3111**	2	**-3111**	2	-3346	2	-3535	2	-3409	2
16000	-6121	1	-6128	2	-6439	2	**-6121**	2	**-6121**	1	-6610	2	-7046	2	-6741	2

Table 4.7: Experiment results for network Studfarm running K2 algorithm

	Optimal		OE1		OE1R		OE2		OE2R		OE3		OE3R		Random	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	-9903	76	**-9904**	76	-9987	80	-11091	76	-9929	77	-10167	74	-10287	73	-10415	74
4000	-19282	74	**-19279**	75	-19507	79	-21447	77	-19362	75	-19793	71	-20058	71	-20276	74
8000	-37865	74	**-37863**	78	-38340	81	-43053	76	-38032	77	-38845	72	-39381	71	-39894	75
16000	-75065	74	**-75063**	77	-76063	79	-85402	74	-75374	77	-76947	72	-78038	72	-78950	74
32000	-149081	72	**-149079**	74	-151049	79	-170089	75	-149652	75	-152741	75	-154921	70	-156584	73
64000	-297310	72	**-297309**	72	-300877	78	-341601	75	-298441	76	-304391	77	-308895	70	-312487	74

Table 4.8: Experiment results for network Boblo running OS algorithm

	Optimal		OE1		OE1R		OE2		OE2R		OE3		OE3R		Random	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	-9907	11	-10251	9	-10187	8	-11333	9	**-10076**	9	-10482	10	-10337	10	-10700	10
4000	-19284	10	-19892	9	-19964	8	-22409	9	**-19682**	9	-20422	9	-20140	10	-20911	10
8000	-37866	10	**-38400**	10	-39288	8	-43954	9	-38715	9	-40110	9	-39564	10	-41221	10
16000	-75066	10	**-75401**	11	-77953	8	-86958	9	-76848	9	-79503	9	-78411	10	-81578	11
32000	-149082	11	**-14921**8	13	-154813	9	-172395	10	-152659	10	-157900	9	-155629	10	-161614	10
64000	-297311	9	**-297315**	13	-308502	9	-343974	9	-304406	9	-314932	10	-310289	10	-322990	10

Table 4.9: Experiment results for network Boblo running K2 algorithm

	Optimal		OE1		OE1R		OE2		OE2R		OE3		OE3R		Random	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	-20525	5255	-20605	5157	-20553	5104	-20589	6075	-20556	5015	**-20549**	4454	-20589	5714	-20593	4955
4000	-41012	5061	-41152	4811	**-41056**	4897	-41129	5793	-41060	4840	**-41056**	4359	-41115	5539	-41113	4890
8000	-81805	4623	-82009	4906	**-81855**	4658	-81966	5709	-81889	4537	-81874	4078	-81969	5348	-81977	4603
16000	-163313	4471	-163562	4443	-163429	4010	-163524	5501	-163448	4090	**-163401**	3779	-163529	5042	-163537	4289
32000	-326405	3841	-326725	4093	-326557	3785	-326681	5045	-326583	3833	**-326513**	3567	-326667	4743	-326716	4071

Table 4.10: Experiment results for network Boerlage92 running OS algorithm

	Optimal		OE1		OE1R		OE2		OE2R		OE3		OE3R		Random	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	-20853	14	-20646	20	-20875	12	-20832	24	-20906	12	**-20569**	14	-20956	15	-20716	16
4000	-41673	9	-41202	21	-41619	12	-41438	22	-41762	12	**-41078**	14	-41785	16	-41393	16
8000	-83140	8	-82096	20	-83148	11	-82175	22	-83293	12	**-81900**	14	-83253	14	-82427	16
16000	-166277	7	-163700	19	-165439	10	-163898	22	-166272	12	**-163437**	14	-165819	13	-164488	16
32000	-332284	7	-326938	18	-331592	10	-327316	21	-332188	11	**-326570**	14	-330636	14	-328587	15

Table 4.11: Experiment results for network Boerlage92 running K2 algorithm

Optimal		OE1		OE1R		OE2		OE2R		OE3		OE3R		Random	
Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
-19639	203566	-20076	215967	-20224	241356	-20754	243875	**-20047**	229714	-20393	232888	-20949	222871	-20552	214183
-38398	217984	**-38740**	231038	-39240	257050	-40382	269315	-38983	243595	-39605	247656	-40412	238285	-39754	229281
-75942	230657	**-76106**	243976	-77064	270909	-79690	278455	-76740	254931	-77857	260097	-79003	252180	-78067	244182
-150729	242566	**-150733**	257271	-152327	276282	-157297	300789	-151726	269695	-153797	272194	-155188	263795	-154068	255417
-300264	264846	**-300269**	284572	-302420	292633	-311660	310943	-301728	281015	-305467	281903	-306974	276841	-305113	280059

Table 4.12: Experiment results for network Alarm running OS algorithm

	Optimal		OE1		OE1R		OE2		OE2R		OE3		OE3R		Random	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	-19681	131	-20164	184	-20404	223	-21425	284	**-20150**	224	-20489	207	-21104	284	-20738	236
4000	-38462	133	-39118	203	-39500	253	-41512	301	**-39110**	243	-39760	233	-40733	332	-39996	263
8000	-76026	139	-76999	219	-77356	281	-81382	340	**-76936**	268	-78104	267	-79575	365	-78504	299
16000	-150851	141	**-151510**	214	-153021	303	-160960	359	-152009	255	-154247	287	-156137	427	-154635	311
32000	-300425	165	**-301277**	234	-303482	310	-319224	369	-302077	266	-306262	331	-308511	454	-306186	367

Table 4.13: Experiment results for network Alarm running K2 algorithm

Tables 4.6 – 4.13 represent averages of resulting BN score and run time for BNs Studfarm (Table 4.6 and 4.7 for OS and K2 respectively), Boblo (Tables 4.8 and 4.9), Boerlage92 (Tables 4.10 and 4.11) and Alarm (Tables 4.12 and 4.13). The best average score of resulting BN is bolded and better than random average score is underlined for every dataset size. Only OE methods are considered while selecting best and better than random averages, as Optimal ordering obviously (and experiments prove that) is always the best. Although best averages are not underlined, they have shown always to be better than random. There is no *Entries* column in Table 4.12, but it is the same as in Table 4.13.

Some simple statistics about every OE algorithm being best or at least better than random can be found in Tables 4.14 – 4.17. According to this statistics OE1 have been best while running OS (Table 4.14) for most of the times. On the other hand OE2R turned out to be best while running K2 (Table 4.15) for most of the times. While getting best result is important and here is where OE1 and OE2R good in case of OS and K2 respectively, reliability is even more important and this is where OE3 together with OE1R and OE1 together with OE3 is better in case of OS and K2 respectively. So OE3 is obviously the most reliable OE method (out of the three proposed) with only two minor exceptions on Alarm (Tables 4.12 and 4.13 32000 entries cases) for both BN structure learning algorithms considered (OS and K2).

It is really hard to make a conclusion which method is generally the best. OE1 could take this position but it provides exceptionally not the best or even the better than random average score for Boerlage92 network in case of OS. As OE1R average scores are better than OE1 (and even best for datasets of size 4000 and 8000) in Table 4.10, it seems that OE1 method prepares pretty much a reversed ordering compared to the optimal one. So, although OE1 seems to be quite reliable in case of K2 it is not even guaranteed to estimate better than random ordering in case of OS.

At this point OE1 and OE2 assumption experiments from section 4.1 should be remembered. The relation of OE1 being highly effective on BNs Studfarm, Boblo and Alarm and OE1 weight correlating well with ancestor count of variables while both factors failing on Boerlage92 indicate, that the assumption at least for OE1 might be true. On the other hand there is OE2 and there is no notable relation in its case. This could lead two ways: either scoring function suits better than mutual information for ordering estimation or the assumption is wrong. Obviously more experiments are needed to conclude which one (or both) are right.

Entries	**OE1**	**OE1R**	**OE2**	**OE2R**	**OE3**	**OE3R**
2000	2	0	0	1	1	0
4000	3	1	0	0	1	0
8000	3	1	0	0	0	0
16000	3	0	0	0	1	0
32000	2	0	0	0	1	0
Total	**13**	2	0	1	4	0

Table 4.14: Best OE algorithm statistics for OS

Entries	OE1	OE1R	OE2	OE2R	OE3	OE3R
2000	0	0	1	3	1	0
4000	0	0	1	3	1	0
8000	1	0	1	2	1	0
16000	2	0	1	1	1	0
32000	2	0	0	0	1	0
Total	5	0	4	**9**	5	0

Table 4.15: Best OE algorithm statistics for K2

Entries	OE1	OE1R	OE2	OE2R	OE3	OE3R
2000	3	3	2	3	4	2
4000	3	4	1	3	4	1
8000	3	3	1	3	3	2
16000	3	3	1	3	3	2
32000	2	3	1	3	2	2
Total	14	**16**	6	15	**16**	9

Table 4.16: Better than random OE algorithm statistics for OS

4.3 Ordering Estimation Compared to Hill-Climber

While OE1 and OE3 methods have provided best results it is also interesting to see how their improvements on OS and K2 compares with Hill-Climber which has been included in experiments. Tables 4.18 – 4.25 provide the results for BNs Studfarm (Tables 4.6 and 4.7 for OS and K2 respectively), Boblo (Tables 4.8 and 4.9), Boerlage92 (Tables 4.10 and 4.11) and Alarm (Tables 4.12 and 4.13). Best average scores are bolded and and better than HC are underlined.

Obviously the results confirm, that even a generic HC is really hard to beat. In case of Studfarm and Alarm OS have managed to produce better structures while in case of Boblo and Boerlage92 HC has advantage. Anyway experiments confirmed that the complexity of OS is higher having maximum number of parents $k > 2$. OS executes faster for Studfarm and Boblo with $k = 2$, but the time grows exponentially when $k = 4$ for Boerlage92 and Alarm.

Anyway, when OS becomes impractical with $k > 2$, K2 proves to be really very fast. Although K2 average scores are worse than HC, running K2 with OE1 estimated orderings results are close to those got by running it with optimal ordering, which is more or less the limit of K2. Also most of them are above the

Entries	OE1	OE1R	OE2	OE2R	OE3	OE3R
2000	4	3	1	3	4	1
4000	4	3	1	3	4	1
8000	4	3	2	3	4	1
16000	4	3	2	3	4	1
32000	3	2	1	2	2	1
Total	**19**	14	7	14	18	5

Table 4.17: Better than random OE algorithm statistics for K2

results of running K2 with random ordering. Having this in mind, K2 becomes more practical as it is now possible to benefit from its high speed without having an optimal ordering which is usually unknown.

	Optimal		OE1		OE3		Random		HC	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	**-843**	7	-844	6	-870	7	-876	7	-860	16
4000	**-1601**	7	-1602	7	-1642	9	-1667	8	-1628	15
8000	**-3111**	10	-3111	9	-3190	12	-3233	12	-3149	20
16000	**-6121**	8	-6121	7	-6281	10	-6358	10	-6181	23

Table 4.18: OS with OE1 and OE3 compared to HC for Studfarm

	Optimal		OE1		OE3		Random		HC	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	**-844**	1	-856	1	-899	1	-909	1	-860	16
4000	**-1602**	1	-1611	1	-1712	2	-1743	1	-1628	15
8000	**-3111**	2	-3113	2	-3346	2	-3409	2	-3149	20
16000	**-6121**	1	-6128	2	-6610	2	-6741	2	-6181	23

Table 4.19: K2 with OE1 and OE3 compared to HC for Studfarm

	Optimal		OE1		OE3		Random		HC	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	-9903	76	-9904	76	-10167	74	-10415	74	**-9884**	110
4000	-19282	74	-19279	75	-19793	71	-20276	74	**-19269**	124
8000	-37865	74	-37863	78	-38845	72	-39894	75	**-37864**	126
16000	**-75065**	74	-75063	77	-76947	72	-78950	74	-75099	140
32000	-149081	72	-149079	74	-152741	75	-156584	73	**-149077**	161
64000	-297310	72	-297309	72	-304391	77	-312487	74	**-297242**	176

Table 4.20: OS with OE1 and OE3 compared to HC for Boblo

	Optimal		OE1		OE3		Random		HC	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	-9907	11	-10251	9	-10482	10	-10700	10	**-9884**	110
4000	-19284	10	-19892	9	-20422	9	-20911	10	**-19269**	124
8000	-37866	10	-38400	10	-40110	9	-41221	10	**-37864**	126
16000	**-75066**	10	-75401	11	-79503	9	-81578	11	-75099	140
32000	-149082	11	-149218	13	-157900	9	-161614	10	**-149077**	161
64000	-297311	9	-297315	13	-314932	10	-322990	10	**-297242**	176

Table 4.21: K2 with OE1 and OE3 compared to HC for Boblo

	Optimal		OE1		OE3		Random		HC	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	**-20525**	5255	-20605	5157	-20549	4454	-20593	4955	-20558	217
4000	-41012	5061	-41152	4811	-41056	4359	-41113	4890	**-40980**	233
8000	-81805	4623	-82009	4906	-81874	4078	-81977	4603	**-81766**	215
16000	-163313	4471	-163562	4443	-163401	3779	-163537	4289	**-163305**	206
32000	-326405	3841	-326725	4093	-326513	3567	-326716	4071	**-326462**	206

Table 4.22: OS with OE1 and OE3 compared to HC for Boerlage92

	Optimal		OE1		OE3		Random		HC	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	-20853	14	-20646	20	-20569	14	-20716	16	**-20558**	217
4000	-41673	9	-41202	21	-41078	14	-41393	16	**-40980**	233
8000	-83140	8	-82096	20	-81900	14	-82427	16	**-81766**	215
16000	-166277	7	-163700	19	-163437	14	-164488	16	**-163305**	206
32000	-332284	7	-326938	18	-326570	14	-328587	15	**-326462**	206

Table 4.23: K2 with OE1 and OE3 compared to HC for Boerlage92

	Optimal		OE1		OE3		Random		HC	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	**-19639**	203566	-20076	215967	-20393	232888	-20552	214183	-19786	1124
4000	**-38398**	217984	-38740	231038	-39605	247656	-39754	229281	-38622	1181
8000	**-75942**	230657	-76106	243976	-77857	260097	-78067	244182	-76258	1390
16000	**-150729**	242566	-150733	257271	-153797	272194	-154068	255417	-151538	1406
32000	**-300264**	264846	-300269	284572	-305467	281903	-305113	280059	-301346	1536

Table 4.24: OS with OE1 and OE3 compared to HC for Alarm

	Optimal		OE1		OE3		Random		HC	
Entries	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
2000	**-19681**	131	-20164	184	-20489	207	-20738	236	-19786	1124
4000	**-38462**	133	-39118	203	-39760	233	-39996	263	-38622	1181
8000	**-76026**	139	-76999	219	-78104	267	-78504	299	-76258	1390
16000	**-150851**	141	-151510	214	-154247	287	-154635	311	-151538	1406
32000	**-300425**	165	-301277	234	-306262	331	-306186	367	-301346	1536

Table 4.25: K2 with OE1 and OE3 compared to HC for Alarm

Chapter 5

Conclusions

This work targets a certain class of Bayesian network structure learning methods – ordering based or dependant algorithms. As they consider variable ordering at some point, this work assumes that manipulating ordering may have some effects on their results.

The idea of ordering estimation is quite straightforward – to estimate variable ordering that is optimal or close to it. The goal of OE is to improve ordering based or dependant BN structure learning algorithms like OS and K2 used in this work. While K2 requires optimal ordering to run OS started with optimal ordering is expected to benefit from that too (section 3).

Three OE methods, that aim to estimate variable position in BN have been proposed:

1. Scoring function approach OE1 (section 3.1)
2. Mutual information approach OE2 (section 3.2)
3. Dependence tree ([7]) approach OE3 (section 3.3)

The experiments have shown that every one of them have different features, provide different results and have different reasons for that. While OE1 have produced higher score structures in many cases, OE3 proved to be more reliable and OE2 turned out to be unsuccessful. In spite of that, both – OS and K2 – have gained from at least one of the methods as they proved to be better than running algorithms withs random ordering.

While OS have provided better quality structures, the most important result of this work is that K2 being impractical because of the requirement to have optimal ordering can be resurrected for some real work. Using ordering estimation have increased the score of resulting structure and made it possible to benefit from the high speed K2 can offer compared to the most popular score based BN structure learning approach – Hill-climber greedy search.

Although some promissing results have been shown in this paper, even more is yet to discover in order to improve quality of ordering estimation as well as find the ways to incorporate it with other solutions in order to achieve even better results.

Chapter 6

Future Work

Quite a few side directions have been considered during this research. Continuing this work, and important task would be to improve OE algorithm. Although OE1 and OE3 have provided satisfactory results in different ways, one might try to come up with a solution that is both – reliable (like OE3) and provides high score results (like OE1) in most cases.

In spite of that, OE1 and OE3 have a potential. Anyway it is not very clear as experiments with more BNs need to be done. On the other hand K2 being very fast compared to HC can be used to prepare initial structure for HC. The results of HC can be expected to improve as starting with some reasonably approximated structure could help to escape local maxima as well as save some iterations and time.

Although OS have been shown to be inefficient when maximum number of parents is more than 2, the exhaustive parent candidate evaluation, which is $O(n^k)$ complex, could be replaced with some more efficient approach. For example fast K2 could be used to approximate best parent set. This could possibly make OS faster than HC. Of course the question would be if resulting structure quality would be at least as good as HC.

Bibliography

[1] W.-K. W. Andrew Moore. Optimal reinsertion: A new search operator for accelerated and more accurate bayesian network structure learning. In T. Fawcett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning (ICML '03)*, pages 552–559, Menlo Park, California, August 2003. AAAI Press.

[2] R. R. Bouckaert. Probabilistic network construction using the MDL principle. Technical report, Department of Computer Science, Utrecht University, July 1994.

[3] R. R. Bouckaert. *Bayesian belief networks: from inference to construction.* PhD thesis, Utrecht Universiteit, 1995.

[4] A. M. Brigham Anderson. Ad-trees for fast counting and for fast learning of association rules. In *Knowledge Discovery from Databases Conference*, 1998.

[5] R. Castelo and T. Kocka. On inclusion-driven learning of bayesian networks. *The Journal of Machine Learning Research*, 4:527–574, 2003.

[6] D. M. Chickering. Learning equivalence classes of bayesian-network structures. *J. Mach. Learn. Res.*, 2:445–498, 2002.

[7] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[8] G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

[9] N. Friedman and M. Goldszmidt. Sequential update of Bayesian network structure. In *Thirteenth Conf. on Uncertainty in Artificial Intelligence*, pages 165–174, 1997.

[10] N. Friedman, I. Nachman, and D. Pe'er. Learning Bayesian network structure from massive datasets: The "Sparse candidate" algorithm. In *Uncertainty in Artificial Intelligence*, pages 206–215, 1999.

[11] F. V. Jensen. *Bayesian Networks and Decision Graphs.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[12] J. Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society, Vol 7, No. 1*, volume 7, pages 48–50. American Mathematical Society, February 1956.

[13] W. Lam and F. Bacchus. Learning Bayesian belief networks: an approach based on MDL principle. *Computational Intelligence*, 10(3):269–293, 1994.

[14] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics & Data Analysis*, 19(2):191–201, 1995.

[15] P. Munteanu and D. Cau. Efficient score-based learning of equivalence classes of bayesian networks. In *PKDD '00: Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 96–105, London, UK, 2000. Springer-Verlag.

[16] R. W. Robinson. Counting unlabeled acyclic digraphs. In *Australian Conference on Combinational Mathematics*, pages 28–43, 1976.

[17] J. Roure. Incremental methods for Bayesian network structure learning: Thesis. *AI Communications*, 18(1):61–62, 2005.

[18] R. Sangüesa and U. Cortés. Learning causal networks from data: a survey and a new algorithm for recovering possibilistic causal networks. *AI Communications*, 10(1):31–61, 1997.

[19] D. J. Spiegelhalter and S. L. Lauritzen. Sequential updating of conditional probabilities on directed graphical structures.

[20] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search, Second Edition (Adaptive Computation and Machine Learning)*. The MIT Press, January 2001.

[21] H. Steck. *Constraint-based Structural Learning in Bayesian Networks Using Finite Data Sets*. PhD thesis, Technische Universität München, 2001.

[22] M. Teyssier and D. Koller. Ordering-based search: A simple and effective algorithm for learning bayesian networks. In *Proceedings of the Twenty-first Conference on Uncertainty in AI (UAI)*, pages 584–590, Edinburgh, Scottland, UK, July 2005.