UNIVERSITÄT REGENSBURG

Institute of Genomics & Practical Bioinformatics
Master of Science Computational Science

The max-min-hill-climbing algorithm

Report
in practical Bioinformatics

by

Michael Bauer

Matrikelnummer: 152 8558

Tutor:	Dr. Giusi Moffa
Adviser:	Prof. Dr. Rainer Spang
Date:	

Contents

List of Figures

List of Tables

1 Abstract

In this report we present a new implementation of the max-min-hill-climbing algorithm (MMHC) for R, first stated in [TBA]. It combines both: greedy search and constraint-based learning techniques. We will discuss those two methods seperately and how they effect running time. We also want to work out the importance of this algorithm. The main goal of it is to reconstruct Bayesian networks from estimated data. Bayesian networks play a great role in science, economics, sports and many more fields where the observational data can get extremely big. It is not the first time R provides this algorithm but we come up by using RCPP (C++ interface for R) for our implementation to have a better chance to deal with big data. Since running time is getting more important we tried to improve it for this algorithm and beat the existing one.

2 Introduction

The max-min-hill-climbing algorithm is one of the state of the art algorithms in statistical computing. The primal aim of this algorithm is reconstructing BN out of estimated data. A Bayesian network is a Directed Acyclic Graph (DAG) whose nodes are random variables and eges represent conditional dependencies. If two random variables are connected they are said to be dependent. If there is no connection they are said to be conditional independent. BNs are more important than one can imagine. They play a great role in everdays life. For example [NBBCW] use them to predict the effect of missense mutations on the protein function. But not only medical science uses Bayesian networks. Another example where scientists used them was football. In [PKA] they refer to an article where european football clubs tried to predict injuries of their players depending on BNs. With a simple Google search you may also find the prediction of stock exchanges and many more. Wikipedia provides a simple but descriptive example which illustrates BNs in a smooth way.

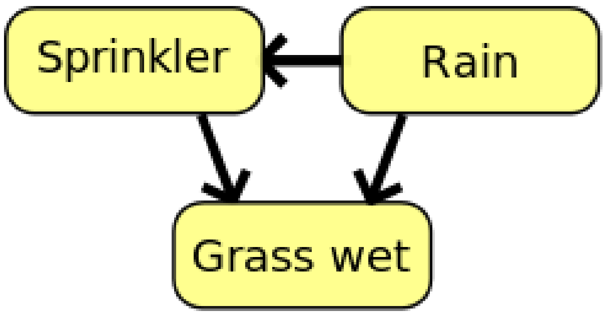


Figure 2.1: A simple example of a Bayesian network.

Reconstructing those networks is not easy, more precisely it is a np-hard problem. It is not only the amount of data which leads to a bad running time, also the dependencies between nodes can slow the code down.
In the first step of this algorithm we try to reconstruct the skeleteon of the graph. Therefor we iterate over all variables - we select one variable (we call it "target" **T**) in each iteration step - and find those variables which are dependent to the selected one. The more variables we find the longer a single iteration takes. The dependent variables are then said to be a parent or a child of **T**. It may happen that there are false positive ones in our set (which we will call

CPC). For this reason we have to check again if the parents/children in the set really belong to our selected variable **T**. The relation between **T** and its parents/children is symmetric. That means for a target **T** and $\mathbf{A} \in \mathbf{CPC}_T$ it follows:

$$\mathbf{T} \longleftrightarrow \mathbf{A} \Longleftrightarrow \mathbf{A} \longleftrightarrow \mathbf{T} \tag{2.1}$$

for a target **A** and $\mathbf{T} \in \mathbf{CPC}_A$. Since this relation holds we have to check in a second step if this symmetrie is fullfild. For that we check for every $\mathbf{X} \in \mathbf{CPC}_T$ if $\mathbf{T} \in \mathbf{CPC}_X$.
Though this is a great approach, we are interested in the whole Directed Acyclic Graph. The second part of the algorithm will then take this skeleton and add directed edges to it such that it does not become cyclic and is fully directed. We will see that this is based on one formula which is not complicated to understand but extremely tricky for implementation. Once we observe the Bayesian network from our data we then can look at the running time of the algorithm and where the time gets lost but also where we optimized to save time. But more important for us was to beat the existing algorithm for R (part of the "bnlearn" package). The goal for us was to be faster. So after a brief discussion of our implementation we will see if it was possible to optimize the code with RCPP to beat the "bnlearn" algorithm.

3 Background

Before we are able to analyze our implementation and talk about it in detail we need some mathematical background. In this section we fully follow [TBA][p. 5-7]. For the proofs of the Lemmas and Theorems we also reference to this paper.

3.1 Notation

We introduce our notation which is completely consistent to [TBA].
We denote

1. variables with an upper-case letter (e.g., A, V_i),
2. a state or a value of that variable by the same lower-case letter (e.g., a, v_i),
3. a set of variables by upper-case bold face (e.g., $\mathbf{Z}, \mathbf{Pa}_i$),
4. an assignment of state or value to each variable in the given set with the corresponding lower-case bold-face letter (e.g., $\mathbf{y}, \mathbf{pa}_i$),
5. special sets of variables (e.g. the set of all variables $\mathcal{V}$) with calligraphic fonts.

3.2 Definition (conditional independence)

Two variables X and Y are conditionally independent given $\mathbf{Z}$ with respect to a probability distribution P, denoted as $Ind_P(X;Y|\mathbf{Z})$, if for all $x, y, \mathbf{z}$ where $P(\mathbf{Z} = \mathbf{z}) > 0$,

$$P(X = x, Y = y|\mathbf{Z} = \mathbf{z}) = P(X = x|\mathbf{Z} = \mathbf{z})P(Y = y|\mathbf{Z} = \mathbf{z}) \tag{3.1}$$

or

$$P(X, Y|\mathbf{Z}) = P(X|\mathbf{Z})P(Y|\mathbf{Z}) \tag{3.2}$$

for short. If X, Y are dependent given $\mathbf{Z}$ we denote $Dep_P(X;Y|\mathbf{Z}$.

3.3 Definition (Bayesian network)

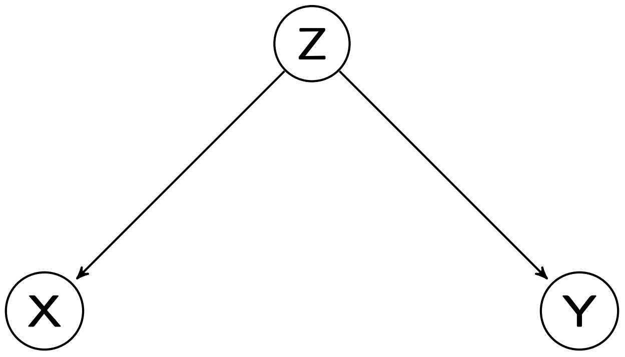
Let P be a discrete joint probability distribution of the random variables in some set $\mathcal{V}$ and $\mathcal{G} =< \mathcal{V}, E >$ be a Directed Acyclic Graph (DAG). We call $< \mathcal{G}, P >$ a (discrete) *Bayesian network* if $< \mathcal{G}, P >$ satisfies the Markov condition.

3.4 Definition (Markov condition)

Any node in a Bayesian network is conditionally independent of its non-descendants, given its parents.

3.5 Explanation

With those definitions we have our first concept we will discuss briefly. We will explain the definitions by using the following picture:



The Markov condition states that X and Y given $\mathbf{Z}$ must be conditionally independent. This comes from the fact that X is a non-descendant of Y and vice versa and $\mathbf{Z}$ is a parent of both nodes. By fullfilling this condition we get from section 3.3 that this graph is a Bayesian network and with section 3.2 we have: $P(X, Y|\mathbf{Z}) = P(X|\mathbf{Z})P(Y|\mathbf{Z})$.

3.6 Defintion (collider)

A node W of a path p is a *collider* if p contains two incomming edges into W.

3.7 Definition (blocked path)

A path p from node X to node Y is *blocked* by a set of nodes $\mathbf{Z}$, if there is a node W on p for which one of the following two conditions hold:

1. W is not a collider and $W \in \mathbf{Z}$, or
2. W is a collider and neither W or its descendants are in $\mathbf{Z}$ [P88]

3.8 Definition (d-seperation)

Two nodes X and Y are *d-seperated* by $\mathbf{Z}$ in graph $\mathcal{G}$ (denoted as $Dsep_{\mathcal{G}}(X;Y|\mathbf{Z})$) if and only if every path from X to Y is blocked by $\mathbf{Z}$. Two nodes are *d-connected* if they are not *d-seperated*.

3.9 Definition (faithful)

If all and only the conditional independencies true in the distribution P are entailed by the Markov condition applied to $\mathcal{G}$, we will say that P and $\mathcal{G}$ are *faithful to each other* ([SGSN]). Furthermore, a distribution P is *faithful* if there exists a graph $\mathcal{G}$, to which it is faithful.

3.10 Definition (faithfulness condition)

A Bayesian network $< \mathcal{G}, P >$ satisfies the *faithfulness condition* if P embodies only independencies that can be represented in the DAG $\mathcal{G}$ ([SGSN]). We will call such a Bayesian network a *faithful network*.

3.11 Theorem

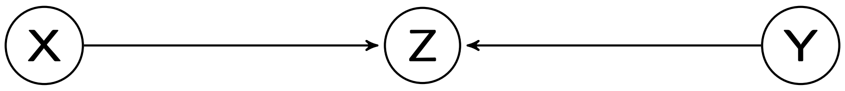
In a faithful Bayesian network $< \mathcal{G}, P >$ the following equivalence holds ([P88])

$$Dsep_{\mathcal{G}}(X;Y|\mathbf{Z}) \Longleftrightarrow Ind_P(X;Y|\mathbf{Z}) \tag{3.3}$$

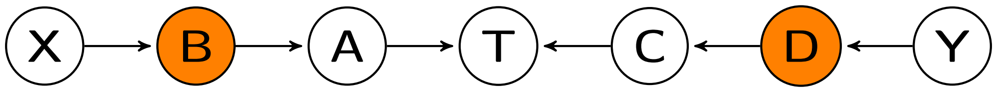
3.12 Remark and Explanation

Remark: For the rest of this report we assume faithfulness of the networks to learn. For this reason we don't want to explain the corresponding definitions in detail. Just note, that the definitions are neccessary for mathematical correctness.

Explanation: The definition of a collider already tells everything about it. To illustrate a collider, we have:



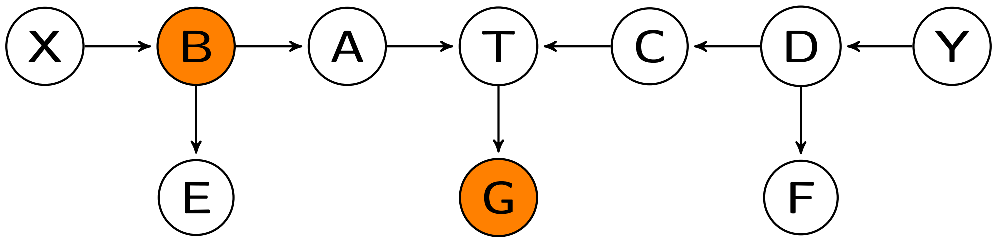
Here Z is a *collider* because it has two incoming edges. In this case if we just look for $P(X;Y|\{\})$, the path between X and Y would be blocked and for this X and Y are *d-seperated*. If we look for $P(X;Y|\mathbf{Z})$, then this path is not blocked and we state that X and Y are *d-connected*.
Because of section 3.11 and the faithfulness assumptions we state for the rest of our report that the terms d-seperation and conditional independence are equivalent. With this we already know that X and Y are conditional dependent given $\mathbf{Z}$ in the example above. This brings us a big step closer to learn the structure of a Bayesian network from observational data. Before we start looking at the algorithms, we want to give you two other examples for d-seperation of variables.



If we are looking for $Ind_P(X;Y|\mathbf{Z})$ with $\mathbf{Z} = \{B, D\}$ we learn that X and Y are conditionally independent given $\mathbf{Z}$. In other words they are d-seperated in the path because of the following reasons:

- T is a collider with $T \notin \mathbf{Z}$ and blocks the path between X and Y.
- The nodes B and D are no colliders but they are elements of $\mathbf{Z}$.

The situation becomes a bit more difficult if we take a look at the next example:



We learn that the path between X and Y remains blocked by looking for $Ind_P(X;Y|\mathbf{Z})$ with $\mathbf{Z} = \{B, G\}$, i.e. X and Y are conditionally independent given $\mathbf{Z}$. If we would look at the path between A and Y we would learn that A and Y are d-connected. This comes from:

- T is a collider but its descendant $G \in \mathbf{Z}$, i.e. T would not block the path.
- The node B is no collider but it is an element of $\mathbf{Z}$. For that it blocks the path.

So there is no element which blocks the path between A and Y, but for X and Y, B blocks the path.
As we could see, detecting conditional independence of two nodes is quite difficult in small graphs. Since we normally observe large data sets with a couple of nodes, a concept for this is needed. As we will see, statistical methods, such as hypothesis testing will be a useful friend for this task.

3.13 Definition

We define the minimum association of X and T relative to a feature subset $\mathbf{Z}$, denoted as $MinAssoc(X;T|\mathbf{Z})$, as

$$MinAssoc(X;T|\mathbf{Z}) = \min_{\mathbf{S} \subseteq \mathbf{Z}} Assoc(X;T|\mathbf{S}) \tag{3.4}$$

i.e., as the minimum association achieved between X and T over all subsets of $\mathbf{Z}$.
Remark:

1. In Figure 5.1.1 we will see this *MinAssoc* statement again as a function. With that we measure conditional independence of X and T given $\mathbf{Z}$.
2. For that the following equivalence holds (providing without a proof):

$$Ind(X;T|\mathbf{Z}) \iff (Assoc(X;T|\mathbf{Z} = 0). \tag{3.5}$$

4 Functions of bnlearn

As we stated, our purpose was to implement the max-min-hill-climbing algorithm in a way that it will be faster and more efficient than the existing is. That's why we want to give you a brief introduction to the mmpc() and mmhc() functions of the "bnlearn" package. We also want to present some numbers depending on running time which show you the effectiveness of the two algorithms. Afterwards we explain our algorithm and we present our results. You will see that there will be some differences on the running time of both implementations.

4.1 mmpc(data.frame)

This function represents the first part of the algorithm, which returns the skeleton of the graph. You can choose between several methods of independence testing. The input is a R data frame and the return value of it is of the class "bn". With the plot function you are able to plot the whole skeleton of the graph, like:

```
> graph <- mmpc(data.frame)
> plot(graph)
```

4.2 mmhc(data.frame)

This function is similiar to the mmpc(). The differences are, that it returns a DAG and you can additionally choose a score function which is used to direct the edges. Again you execute this with:

```
> graph <- mmhc(data.frame)
> plot(graph)
```

5 The max-min-hill-climbing algorithm

We now want to go into the implementation of our algorithm. First of all we want you present the pseudo code and then talk about it. Afterwards we present the statistical methods behind the algorithm and how they effect running time. In the end we want to present an example which should be reconstructed by our algorithm.

5.1 Pseudo code MMHC

```
Algorithm  MMHC Algorithm
1: procedure MMHC($\mathcal{D}$)
       Input: data $\mathcal{D}$
       Output: a DAG on the variables in $\mathcal{D}$
       % Restrict
2:     for every variable $X \in \mathcal{V}$ do
3:         $\mathbf{PC}_X = \text{MMPC}(X, \mathcal{D})$
4:     end for
       % Search
5:     Starting from an empty graph perform Greedy Hill-Climbing
           with operators add-edge, delete-edge, reverse-edge. Only try
           operator add-edge $Y \rightarrow X$ if $Y \in \mathbf{PC}_X$.
6:     Return the highest scoring DAG found
7: end procedure
```

Figure 5.1: The pseudo code of the MMHC algorithm. Line 2-4 represent the loop over all nodes calling the MMPC function. At line 5 the scoring starts.

As mentioned before, the MMHC function has to parts. First we find the skeleton and then direct the edges of the skeleton graph. We first discuss the *MMPC* (max-min parents and children) function which is called the observational data $\mathcal{D}$ in the for loop (over all possible nodes in the graph) and then we take a closer look at the scoring function starting at line 5. This function will later be stated as *BDeu* which stands for Bayesian Dirichlet likelihood-equivalence uniform.

5.1.1 max-min parents and children (MMPC))

The max-min parents and children (MMPC) is the algorithm which reconstructs the graph skeleton from data. To bring you the concept of this closer, let us first present the pseudo code of it:

```
Algorithm  Algorithm MMPC
1: procedure MMPC(T,$\mathcal{D}$)
2:     CPC = $\overline{MMPC}(T, \mathcal{D})$
3:     for every variable $X \in$ CPC do
4:         if $T \notin \overline{MMPC}(X, \mathcal{D})$ then
5:             CPC = CPC $\setminus X$
6:         end if
7:     end for

8:     return CPC
9: end procedure
```

Figure 5.2: The pseudo code of the MMPC function. First the **CPC** set is computed by the $\overline{MMPC}$ function. Afterwards the false positives are erased.

In this algorithm another function $\overline{MMPC}$ is executed. In Figure 5.1.1 this function - including the *MaxMinHeuristic* function - is shown. Back to Figure 5.1.1, we see, that $\overline{MMPC}$ is executed twice. Firstly at line 2 and secondly in the if-statement at line 4. As we stated in Equation 2.1, for two variables they are connected, the symmetrie holds. At line 2 we only now that some X are in the set **CPC** for a target variable T, i.e. the X is a parent or a child of T. By testing if T is also a parent or child of X, we see if the symmetrie holds, if not, X is removed from the **CPC** set. Let's talk about the $\overline{MMPC}$ function. First take a look at is:

Algorithm $\overline{MMPC}$ Algorithm

```
1:  procedure $\overline{MMPC}$ $(T,\mathcal{D})$
        Input: target variable $T$; data $\mathcal{D}$
        Output: the parents and children of $T$ in any Bayesian
        network faithfully representing the data distribution
        %Phase I: Forward
2:      $\mathbf{CPC} = \emptyset$
3:      repeat
4:          $\langle F, assocF \rangle = MaxMinHeuristic(T; \mathbf{CPC})$
5:          if $assocF \neq 0$ then
6:              $\mathbf{CPC} = \mathbf{CPC} \cup F$
7:          end if
8:      until CPC has not changed

        %Phase II: Backward
9:      for all $X \in \mathbf{CPC}$ do
10:         if $\exists \mathbf{S} \subseteq \mathbf{CPC}$, s.t. $Ind(X;T|\mathbf{S})$ then
11:             $\mathbf{CPC} = \mathbf{CPC} \setminus \{X\}$
12:         end if
13:     end for

14:     return CPC
15: end procedure

16: procedure MaxMinHeuristic$(T,\mathbf{CPC})$
        Input: target variable $T$; subset of variables CPC
        Output: the maximum over all variables of the minimum asso-
        ciation with $T$ relative to CPC, and the variable that achieves
        the maximum
17:     $assocF = \max_{X \in \mathcal{V}} MinAssoc(X;T|\mathbf{CPC})$
18:     $F = \arg\max_{X \in \mathcal{V}} MinAssoc(X;T|\mathbf{CPC})$
19:     return $\langle F, assocF \rangle$
20: end procedure
```

Figure 5.3: Here you see both: the $\overline{MMPC}$ function and the calculation of the association between X and T by the *MaxMinHeuristic* function.

In this subfunction we have two important routines: the execution of $\overline{MMPC}$ and the *MaxMinHeuristic*.

5.1.1.1 The *MaxMinHeuristic* function

This function finds the X which maximizes the measure of association between X and T given the current **CPC** set. The current **CPC** set is therefor passed into the function. The return of the function is the X and the value of association between X and T given

CPC. At this point, we left one important question: How do we measure this association.

5.1.1.2 The G^2 value

Because this algorithm is based on conditional independence testing and measuring the strength of association between two variables, we need some formulas for implementation to get this measure. We followed [SGSN] and calculated the G^2 statistic, under the null hypothesis of the conditional independence holding. For that we have (from [TBA]):
Let $S_{ijk}^{ab\mathbf{c}}$ be the number of times in the data where $X_i = a$, $X_j = b$ and $X_k = \mathbf{c}$. We define in a similar fashion, $S_{ik}^{a\mathbf{c}}$, $S_{jk}^{b\mathbf{c}}$ and $S_k^{\mathbf{c}}$, then the G^2 statistic is defined as (c.f. [SGSN]):

$$G^2 := 2 * \sum_{a,b,c} S_{ijk}^{ab\mathbf{c}} * ln\left(\frac{S_{ijk}^{ab\mathbf{c}} * S_k^{\mathbf{c}}}{S_{ik}^{a\mathbf{c}} * S_{jk}^{b\mathbf{c}}}\right), \tag{5.1}$$

The G^2 statistic is asymptotically distributed as χ^2 with appropriate degrees of freedom. To compute these degrees of freedom we use:

$$df = (|D(X_i)| - 1)(|D(X_j)| - 1) \prod_{X_l \in \mathbf{X_k}} |D(X_l)|, \tag{5.2}$$

where $D(X_i)$ is the domain (number of distinct values) of variable X_i. After bringing this to code, we can work with the output of this function.

5.1.1.3 How $\overline{MMPC}$ works

The procedure of the $\overline{MMPC}$ function then works as follows:

- Compute the G^2 value. You observe a *Svalue* of type *double* value.
- Compute the degrees of freedom df.
- Assign the *pvalue* to the output of the function $pchisq(Svalue, df)$.
- Test if the *pvalue* is smaller than 0.05, if yes then keep this *pvalue*, it's corresponding *Svalue* and X.
- At the end let the X join the **CPC** set which has the smallest *pvalue*.
- If two pvalues are equaled then take the variable X with the higher *Svalue*.
- If you checked all variables over all subsets of the **CPC** set, stop the calculations.

Hint: At the moment you find that a target T and a variable X are conditionally independent, i.e. the association values equales zero, you don't need to check this connection again since you already know of their independence. The same holds for the fact, if they are dependent. This saves running time.

Bibliography

[TBA] Ioannis Tsamardinos, Laura E. Brown, Constantin F. Aliferis,
The max-min hill-climbing Bayesian network structure learning algorithm,
Springer Science + Business Media, Inc. 2006

[NBBCW] Chris J. Needham1, James R. Bradford, Andrew J. Bulpitt, Matthew A. Care and David R. Westhead,
Predicting the effect of missense mutations on protein function: analysis with Bayesian networks,
http://www.biomedcentral.com/1471-2105/7/405

[PKA] http://www-ekp.physik.uni-karlsruhe.de/županc/WS1011/docs/Datenanalyse2010_3.pdf

[P88] Pearl, 1988

[SGSN] Spirtes, Glymour & Scheines, 1993, 2000;
Neapolitan, 2003