Learning the *topology* of Bayesian Networks using the *K2* algorithm

Group: *Città Romanze*

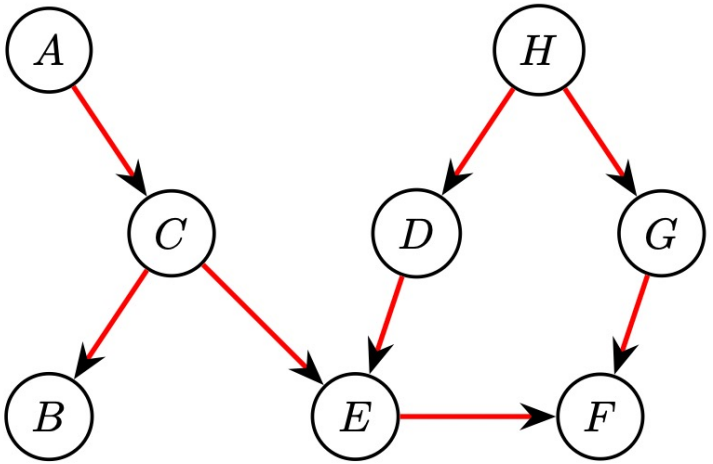
- Giacomo Franceschetto
- Nicolò Lai

Bayesian Networks

A gentle introduction to ***Bayesian Networks*** *(BNs)*

Theoretical introduction

- Directed Acyclic Graph (*DAG*)
- **Nodes**: domain variables
- **Arcs**: probabilistic dependencies



Theoretical introduction

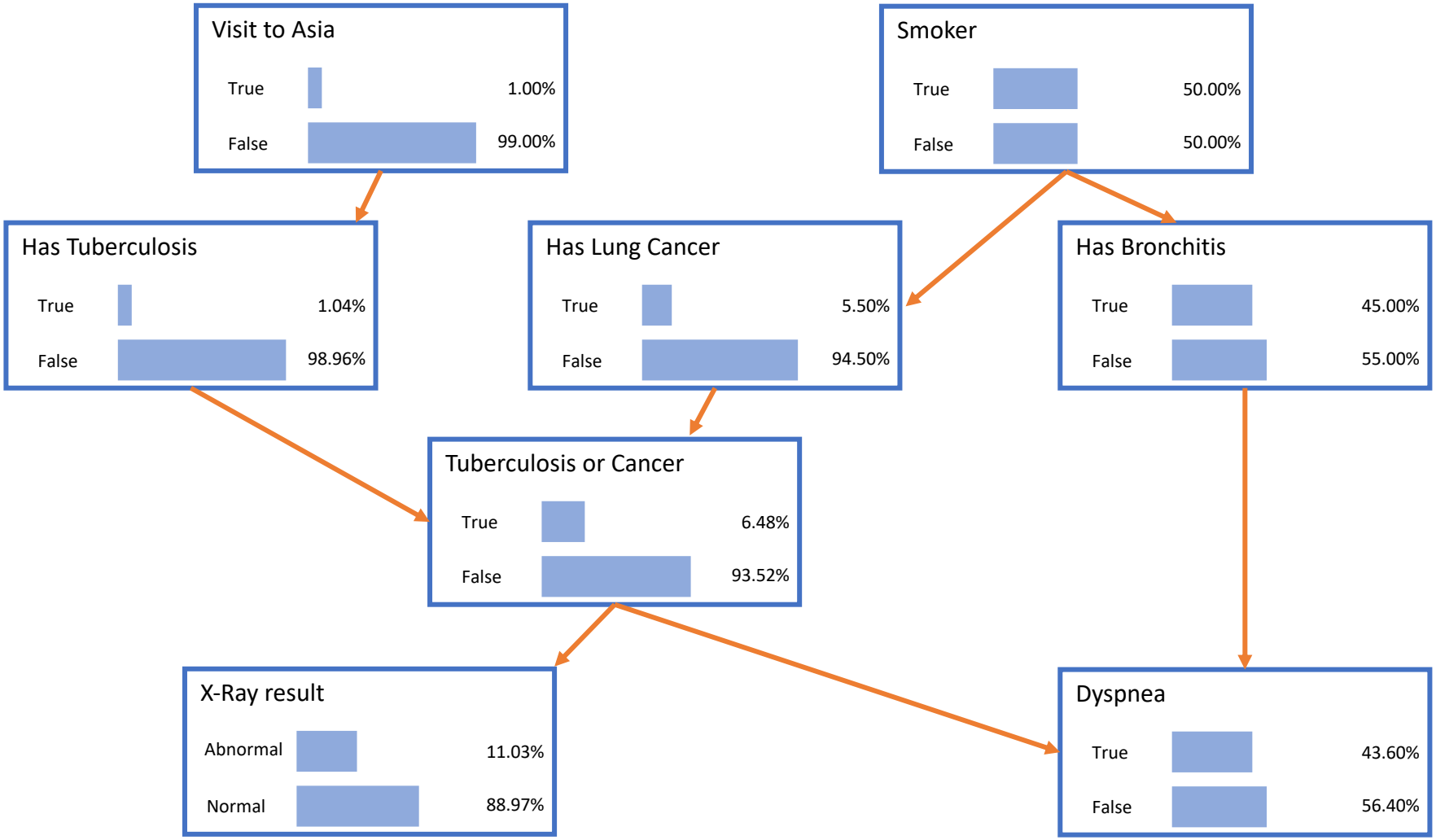
- There is a conditional probability function that relates each **nodes** to its **parents**

$$P(X_i|\pi_i)$$

- **Key feature**: explicit representation of conditional **independence** / **dependence** among events

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\pi_i)$$

ASIA Medical Diagnosis System



Learning Bayesian Networks

A Bayesian Network is a couple $B = (B_s, B_p)$

The ***network probability***, given the data, is

$$P(B|D) = P(B_s, B_p|D) = P(B_s|D) \times P(B_p|B_s, D)$$

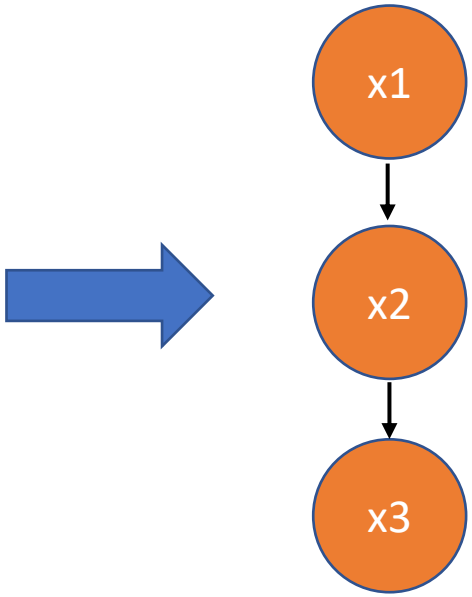
Structure learning

Parameter learning

Learning the Network Structure

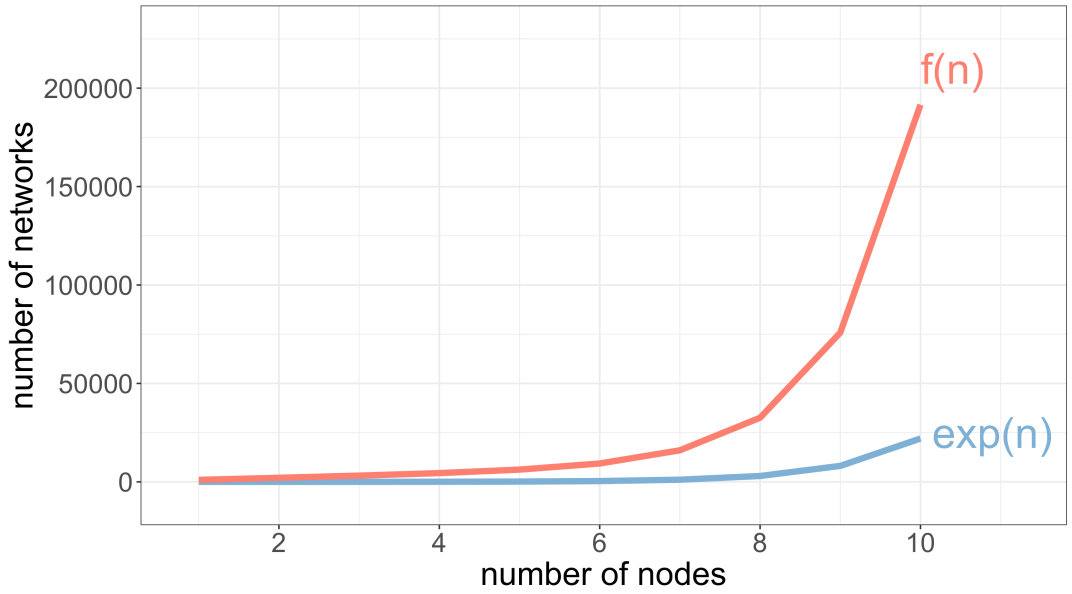
Find the most probable ***Bayesian Network*** ***structure*** given a database

x1	x2	x3
1	0	0
1	1	1
0	0	1
1	1	1
…	…	…



Learning the Network Structure

- **Difficult** and **time-consuming** task
- The **number of possible networks** made of *n* nodes grows super-exponentially

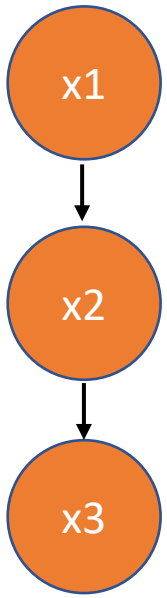


Example with $n = 3$ nodes

(D)

x1	x2	x3
1	0	0
1	1	1
…	…	…

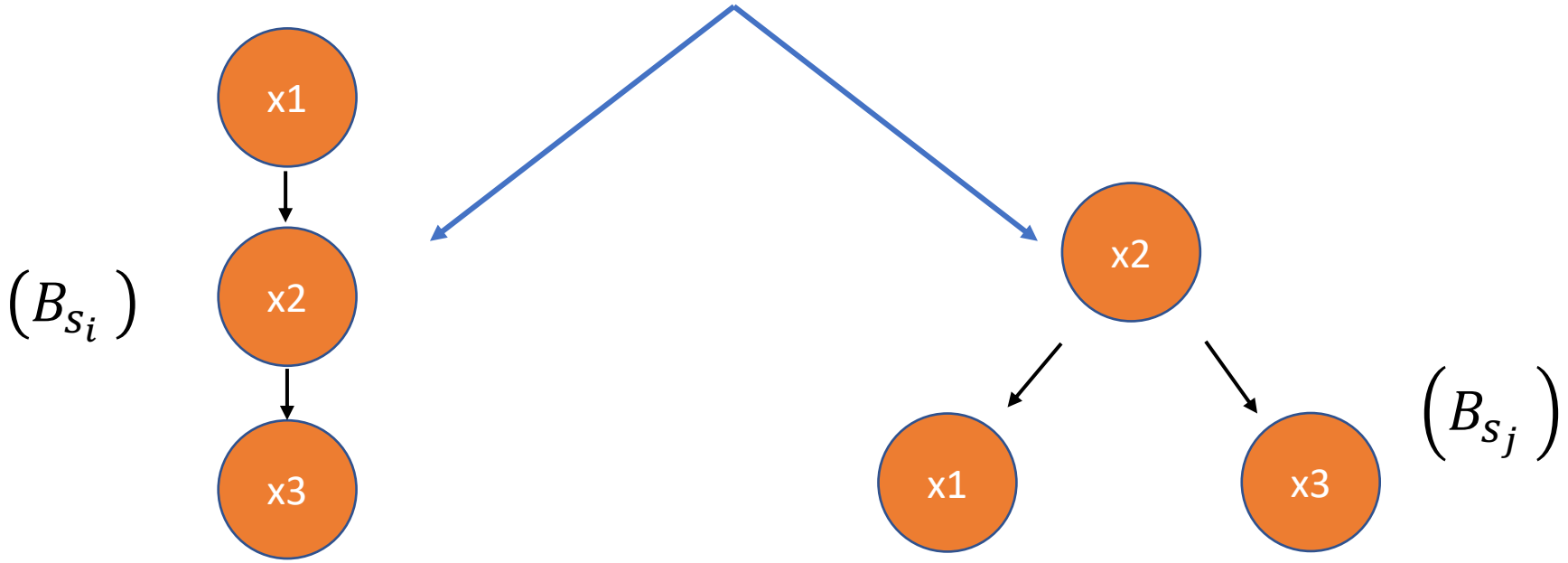
(B_{S_i})



Example with $n = 3$ nodes

(D)

x1	x2	x3
1	0	0
1	1	1
…	…	…



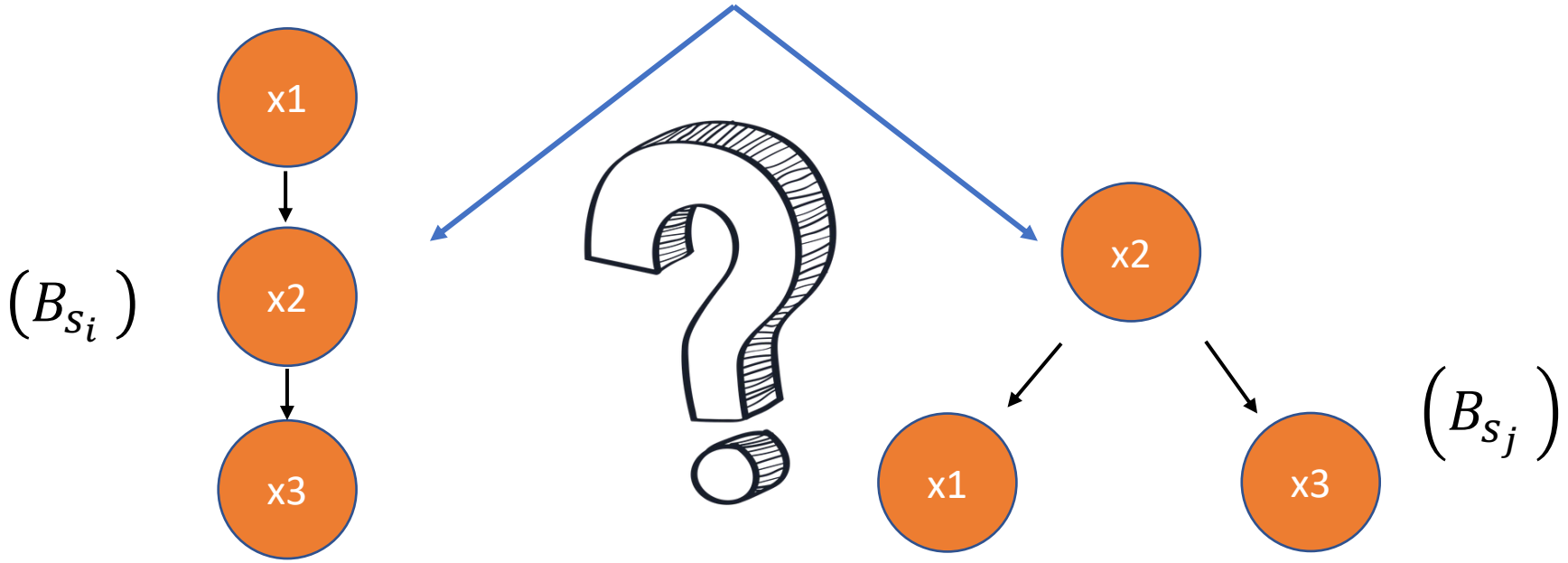
(B_{s_i})

(B_{s_j})

Example with $n = 3$ nodes

(D)

x1	x2	x3
1	0	0
1	1	1
…	…	…



(B_{s_i})

(B_{s_j})

In formulas . . .

Given two possible structures B_{s_i} and B_{s_j}

$$\frac{P(B_{s_i}|D)}{P(B_{s_j}|D)} = \frac{P(B_{s_i}, D)}{P(B_{s_i}, D)}$$

In formulas . . .

Given two possible structures B_{s_i} and B_{s_j}

$$\frac{P(B_{s_i}|D)}{P(B_{s_j}|D)} = \frac{P(B_{s_i}, D)}{P(B_{s_i}, D)}$$

$$P(B_s, D) = ?$$

Finding the probability $P(B_s, D)$

Assumptions:

Finding the probability $P(B_s, D)$

Assumptions:

1. All domain variables are *discrete*

Finding the probability $P(B_s, D)$

Assumptions:

1. All domain variables are discrete
2. Data records are *independent* from each other

Finding the probability $P(B_s, D)$

Assumptions:

1. All domain variables are discrete
2. Data records are independent from each other
3. There are no records with *missing values*

Finding the probability $P(B_s, D)$

Assumptions:

1. All domain variables are discrete
2. Data records are independent from each other
3. There are no records with missing values
4. The *probability density function* of the ***conditional probabilities*** given the network ***structure*** is *uniform*

Finding the probability $P(B_s, D)$

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

where:

- n is the number of variables
- Each variable x_i has r_i possible values assignments $\{v_{ik}\}$
- There are q_i unique instantiations $\{w_{ij}\}$ of x_i parents
- N_{ijk} is the number of cases in which $x_i = \{v_{ik}\}$ and their parents are instantiated as w_{ij}
- Finally: $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$

K2 Algorithm

A ***score-based*** method for maximizing $P(B_S, D)$

Prerequisites

We make the following assumptions:

1. Domain variables are ***ordered***
2. A priori, all structures are equally likely

Now, we reformulate $P(B_s, D)$ as:

$$P(B_s|D) \propto \prod_{i=1}^{n} f(i, \pi_i)$$

with

$$f(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \quad (20)$$

Algorithm *pseudo-code*

```
1.  procedure K2;
2.  {Input: A set of n nodes, an ordering on the nodes, an upper bound u on the
3.          number of parents a node may have, and a database D containing m cases.}
4.  {Output: For each node, a printout of the parents of the node.}
5.  for i:= 1 to n do
6.      π_i := ∅;
7.      P_old := f(i, π_i); {This function is computed using Equation 20.}
8.      OKToProceed := true;
9.      While OKToProceed and |π_i| < u do
10.             let z be the node in Pred(x_i) - π_i that maximizes f(i, π_i ∪ {z});
11.             P_new := f(i, π_i ∪ {z});
12.             if P_new > P_old then
13.                     P_old := P_new;
14.                     π_i := π_i ∪ {z};
15.             else OKToProceed := false;
16.     end {while};
17.     write('Node: ', x_i, ' Parent of x_i: ',π_i);
18. end {for};
19. end {K2};
```

Algorithm implementation in **R**

```
K2 = function(df, u=length(colnames(df))-1, verbose=TRUE, vis=TRUE){
    ######################## K2 algorithm ( df has to be ordered! ) ######################
    # compute all possible instantiations
    inst = lapply(df, n_distinct)
    r    = to_vec(for (i in 1:length(inst)) inst[[i]])
    # define empty dag
    b_s = empty.graph(nodes=colnames(df))
    # K2 score
    k2_score = 0
    # compute parents for each node
    for (i in 1:length(colnames(df))){
        parents_i = vector()
        p_old = log_f(i, parents_i, r[i], df) # compute equation 20
        ok_to_proceed = ifelse(i==1, FALSE, TRUE) # if i==1, then no parents are possible
        pred_i = seq(1, i-1)
        while (ok_to_proceed && length(parents_i)<u && length(pred_i)-length(parents_i)>0){
            z = best_new_parent(i, parents_i, pred_i, r[i], df)
            p_new = log_f(i, c(parents_i, z), r[i], df)
            if (p_new>p_old){
                p_old = p_new
                parents_i = c(parents_i, z)
                b_s = set.arc(b_s, from=colnames(df)[z], to=colnames(df)[i])
            }else{
                ok_to_proceed = FALSE
            }
        }
        k2_score = k2_score + p_old
    }
    return(b_s)
```

Algorithm implementation in R

```
K2 = function(df, u=length(colnames(df))-1, verbose=TRUE, vis=TRUE){
    ######################## K2 algorithm ( df has to be ordered! ) ######################
    # compute all possible instantiations
    inst = lapply(df, n_distinct)
    r    = to_vec(for (i in 1:length(inst)) inst[[i]])
    # define empty dag
    b_s = empty.graph(nodes=colnames(df))
    # K2 score
    k2_score = 0
    # compute parents for each node
    for (i in 1:length(colnames(df))){
        parents_i = vector()
        p_old = log_f(i, parents_i, r[i], df) # compute equation 20
        ok_to_proceed = ifelse(i==1, FALSE, TRUE) # if i==1, then no parents are possible
        pred_i = seq(1, i-1)
        while (ok_to_proceed && length(parents_i)<u && length(pred_i)-length(parents_i)>0){
            z = best_new_parent(i, parents_i, pred_i, r[i], df)
            p_new = log_f(i, c(parents_i, z), r[i], df)
            if (p_new>p_old){
                p_old = p_new
                parents_i = c(parents_i, z)
                b_s = set.arc(b_s, from=colnames(df)[z], to=colnames(df)[i])
            }else{
                ok_to_proceed = FALSE
            }
        }
        k2_score = k2_score + p_old
    }
    return(b_s)
```

Algorithm implementation in R

```
K2 = function(df, u=length(colnames(df))-1, verbose=TRUE, vis=TRUE){
    ######################## K2 algorithm ( df has to be ordered! ) ######################
    # compute all possible instantiations
    inst = lapply(df, n_distinct)
    r    = to_vec(for (i in 1:length(inst)) inst[[i]])
    # define empty dag
    b_s = empty.graph(nodes=colnames(df))
    # K2 score
    k2_score = 0
    # compute parer
    for (i in 1:ler
        parents_i =
        p_old = log
        ok_to_proce                                                    nts are possible
        pred_i = se
        while (ok_t                                                    gth(parents_i)>0){
            z = bes
            p_new =
            if (p_r
                p_c
                par
                b_s                                                    i])
            }else{
                ok_to_proceed = FALSE
            }
        }
        k2_score = k2_score + p_old
    }
    return(b_s)
```

```
# compute eq 20 (log)
log_f = function(i, parents_i, r_i, df){
    # take care of step zero
    if(length(parents_i)==0){
        alpha = table(df[i])
        return(log_f_i_(alpha, r_i))
    }else{
        alpha = table(df[c(parents_i, i)])
        dim(alpha) = c(length(alpha)/r_i, r_i)
        return(log_f_ij(alpha, r_i))
    }
}
```

Algorithm implementation in R

```r
log_f_i_ = function(alpha, r_i){
    N_i_ = sum(alpha)
    return(
            sum(ifelse((r_i-1)==0, 0, sapply(1:(r_i-1),log)))-
            sum(sapply(1:(N_i_+r_i-1),log)) +
            sum(sapply(alpha, function(a_k){
                                sum(ifelse(a_k==0, 0, sapply(1:(a_k),log)))
                            }
                )
            )
    )
}
log_f_ij = function(alpha, r_i){
    N_ij = rowSums(alpha)
    return(
            sum(sapply(1:length(N_ij), function(j){
                                        sum(ifelse((r_i-1)==0, 0, sapply(1:(r_i-1), log))) -
                                        sum(sapply(1:(N_ij[j]+r_i-1), log)) +
                                        sum(sapply(alpha[j, ], function(a_k){
                                                                    ifelse(a_k==0, 0, sum(sapply(1:(a_k),log)))
                                                                }
                                            )
                                        )
                                    }
                )
            )
    )
}
```

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

Algorithm implementation in R

```
K2 = function(df, u=length(colnames(df))-1, verbose=TRUE, vis=TRUE){
    ######################## K2 algorithm ( df has to be ordered! ) ######################
    # compute all possible instantiations
    inst = lapply(df, n_distinct)
    r    = to_vec(for (i in 1:length(inst)) inst[[i]])
    # define empty dag
    b_s = empty.graph(nodes=colnames(df))
    # K2 score
    k2_score = 0
    # compute parents for each node
    for (i in 1:length(colnames(df))){
        parents_i = vector()
        p_old = log_f(i, parents_i, r[i], df) # compute equation 20
        ok_to_proceed = ifelse(i==1, FALSE, TRUE) # if i==1, then no parents are possible
        pred_i = seq(1, i-1)
        while (ok_to_proceed && length(parents_i)<u && length(pred_i)-length(parents_i)>0){
            z = best_new_parent(i, parents_i, pred_i, r[i], df)
            p_new = log_f(i, c(parents_i, z), r[i], df)
            if (p_new>p_old){
                p_old = p_new
                parents_i = c(parents_i, z)
                b_s = set.arc(b_s, from=colnames(df)[z], to=colnames(df)[i])
            }else{
                ok_to_proceed = FALSE
            }
        }
        k2_score = k2_score + p_old
    }
    return(b_s)
```

Algorithm implementation in R

```r
K2 = function(df, u=length(colnames(df))-1, verbose=TRUE, vis=TRUE){
    ######################## K2 algorithm ( df has to be ordered! ) ######################
    # compute all possible instantiations
    inst = lapply(df, n_distinct)
    r    = to_vec(for (i in 1:length(inst)) inst[[i]])
    # define em
    b_s = empty
    # K2 score
    k2_score = 0
    # compute pa
    for (i in 1
        parents_
        p_old =
        ok_to_pr
        pred_i =
        while (o
            z =
            p_ne
            if (
                }els
            }
        }
        k2_score = k2_score + p_old
    }
    return(b_s)
}
```

```r
# compute new parent that maximizes log_f
best_new_parent = function(i, parents_i, pred_i, r_i, df){
    best_score = -Inf
    if(length(pred_i)==1){
        return(pred_i)
    }else{
        z_list = setdiff(pred_i, parents_i)
        for(z in z_list){
            score = log_f(i, c(parents_i, z), r_i, df)
            if(score > best_score){
                best_score = score
                best_z = z
            }
        }
        return(best_z)
    }
}
```

Network *topology* learning

Algorithm tests on *three* different datasets

a) Dummy dataset → *3 nodes*
b) Earthquake → *5 nodes*
c) Child → *20 nodes*

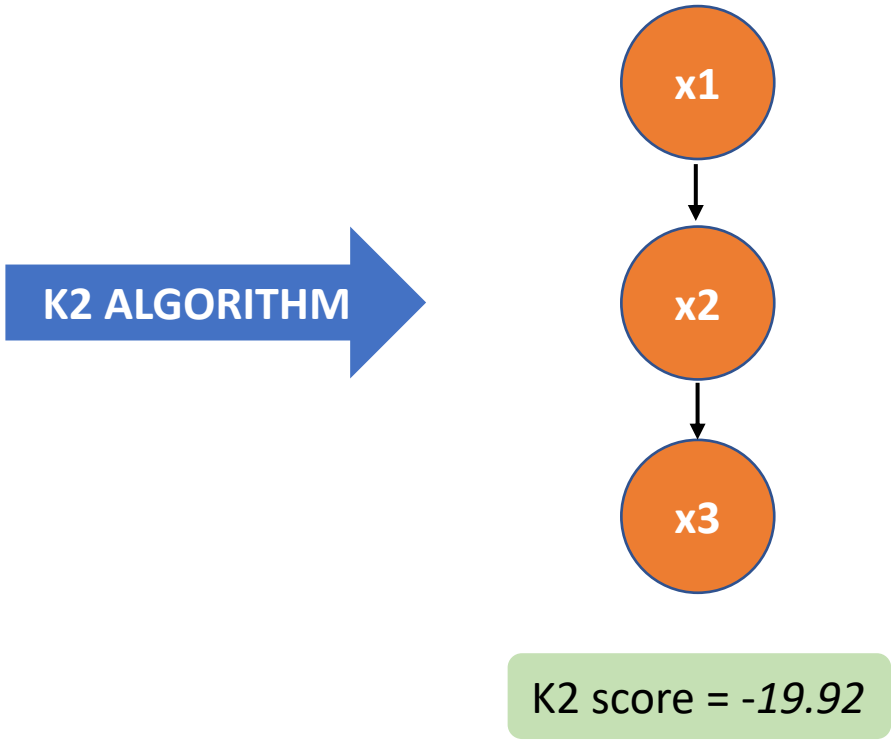
The ***increasing complexity*** of datasets allows to explore the ***K2 algorithm pitfalls***

Dummy Dataset

x1	x2	x3
1	0	0
1	1	1
0	0	1
1	1	1
0	0	0
0	1	1
1	1	1
0	0	0
1	1	1
0	0	0

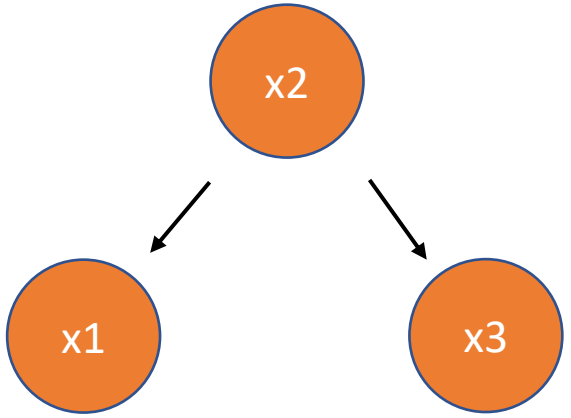
Dummy dataset

x1	x2	x3
1	0	0
1	1	1
0	0	1
1	1	1
0	0	0
0	1	1
1	1	1
0	0	0
1	1	1
0	0	0



Dummy dataset

x1	x2	x3
1	0	0
1	1	1
0	0	1
1	1	1
0	0	0
0	1	1
1	1	1
0	0	0
1	1	1
0	0	0

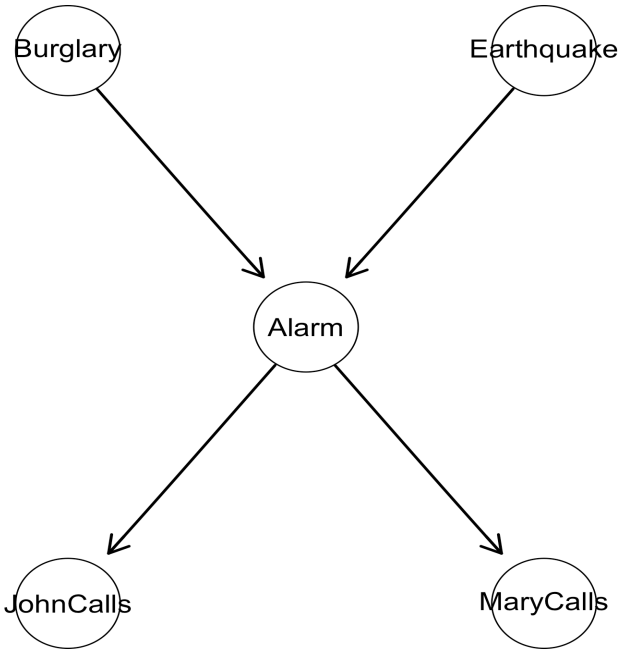


K2 score = *-20.74*

Less probable!

Earthquake dataset

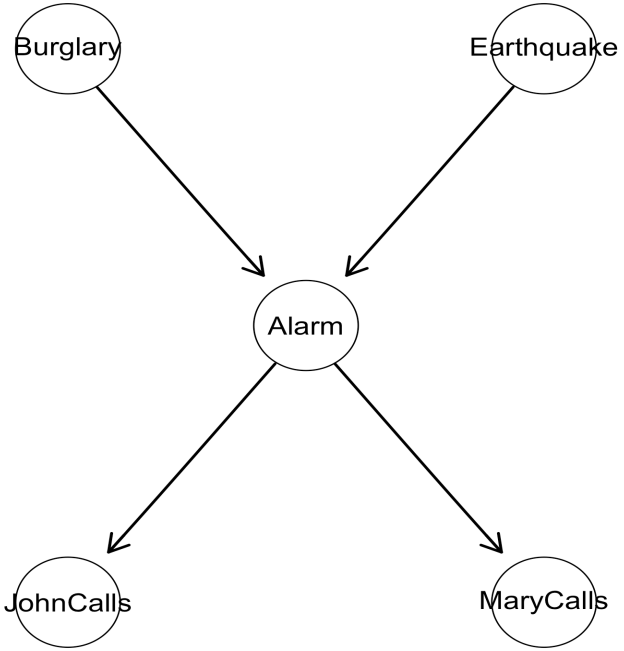
The ***earthquake*** dataset is generated starting from the ***true*** underlying network



link to bnlearn dataset repository

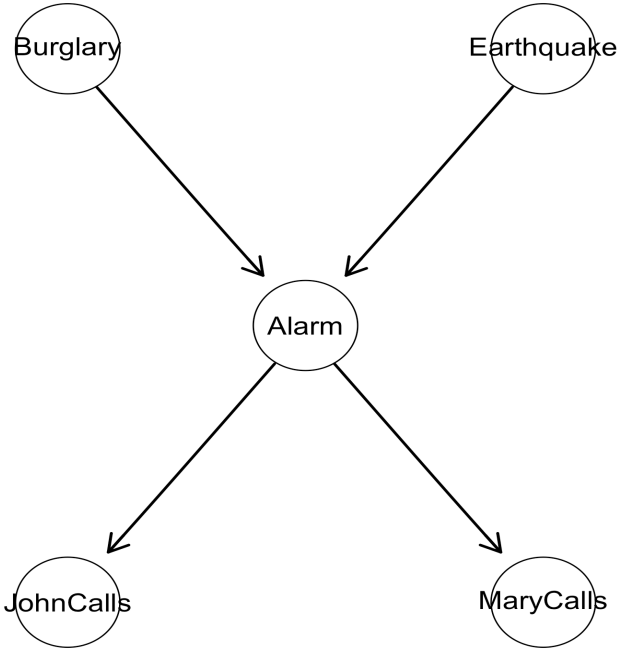
Earthquake dataset

True Network



K2 score = *-441633*

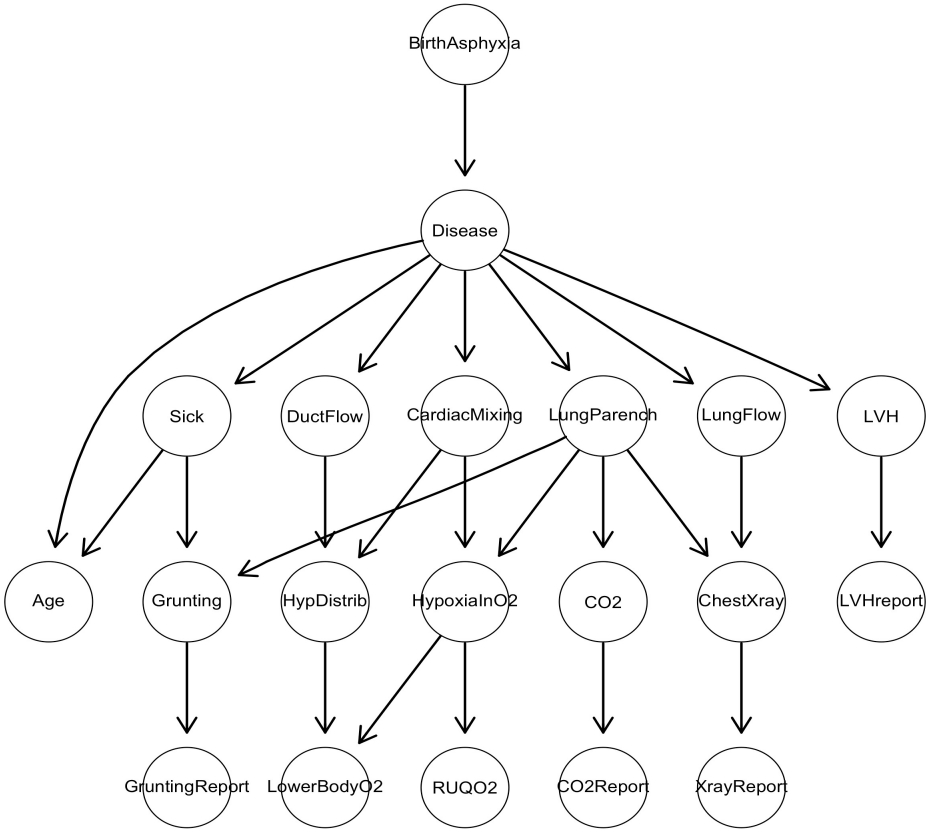
K2 Most Probable Network



K2 score = *-441633*

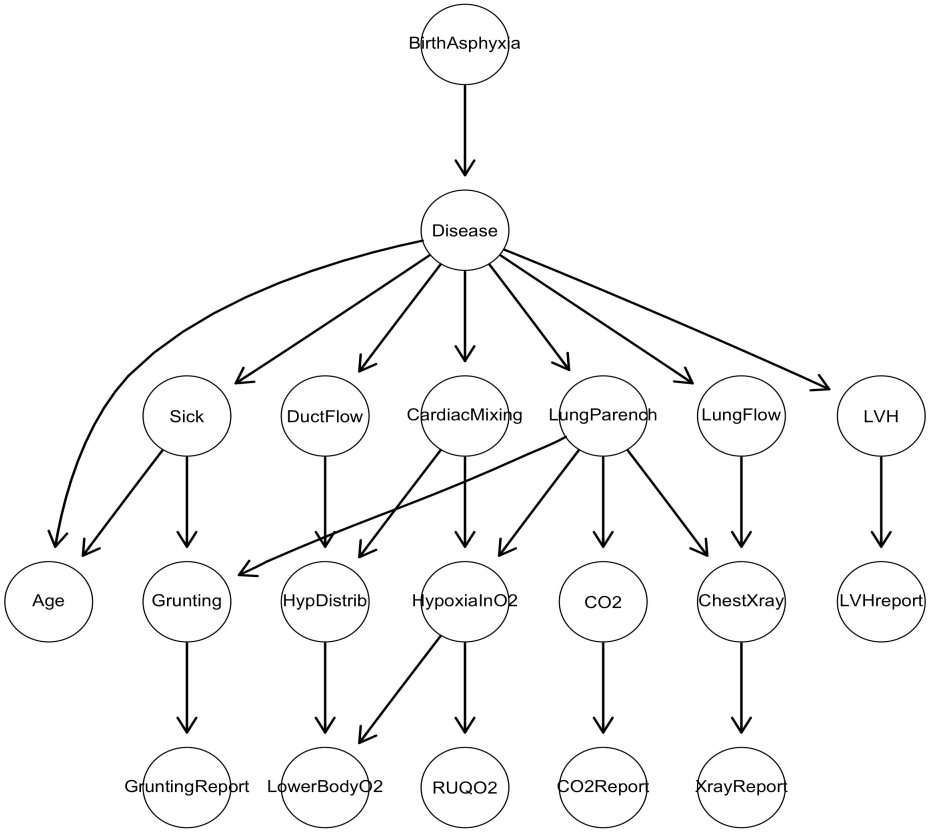
Child dataset

True Network



K2 score = *-610170*

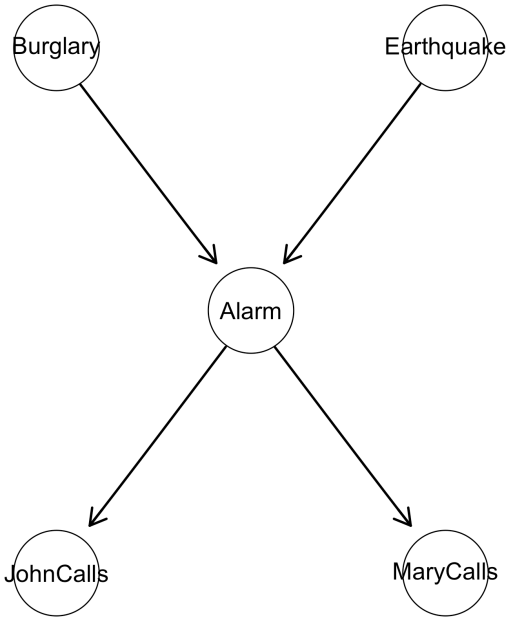
K2 Most Probable Network



K2 score = *-610170*

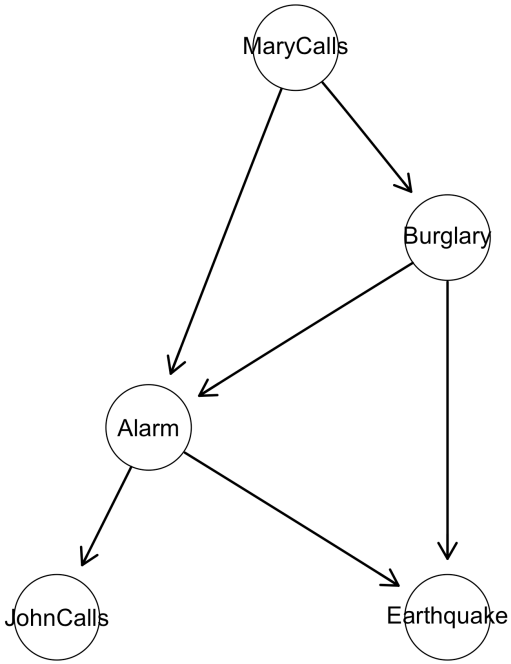
However . . .

The *K2 algorithm* performance *strongly depends* on the initial ***ordering*** of the variables



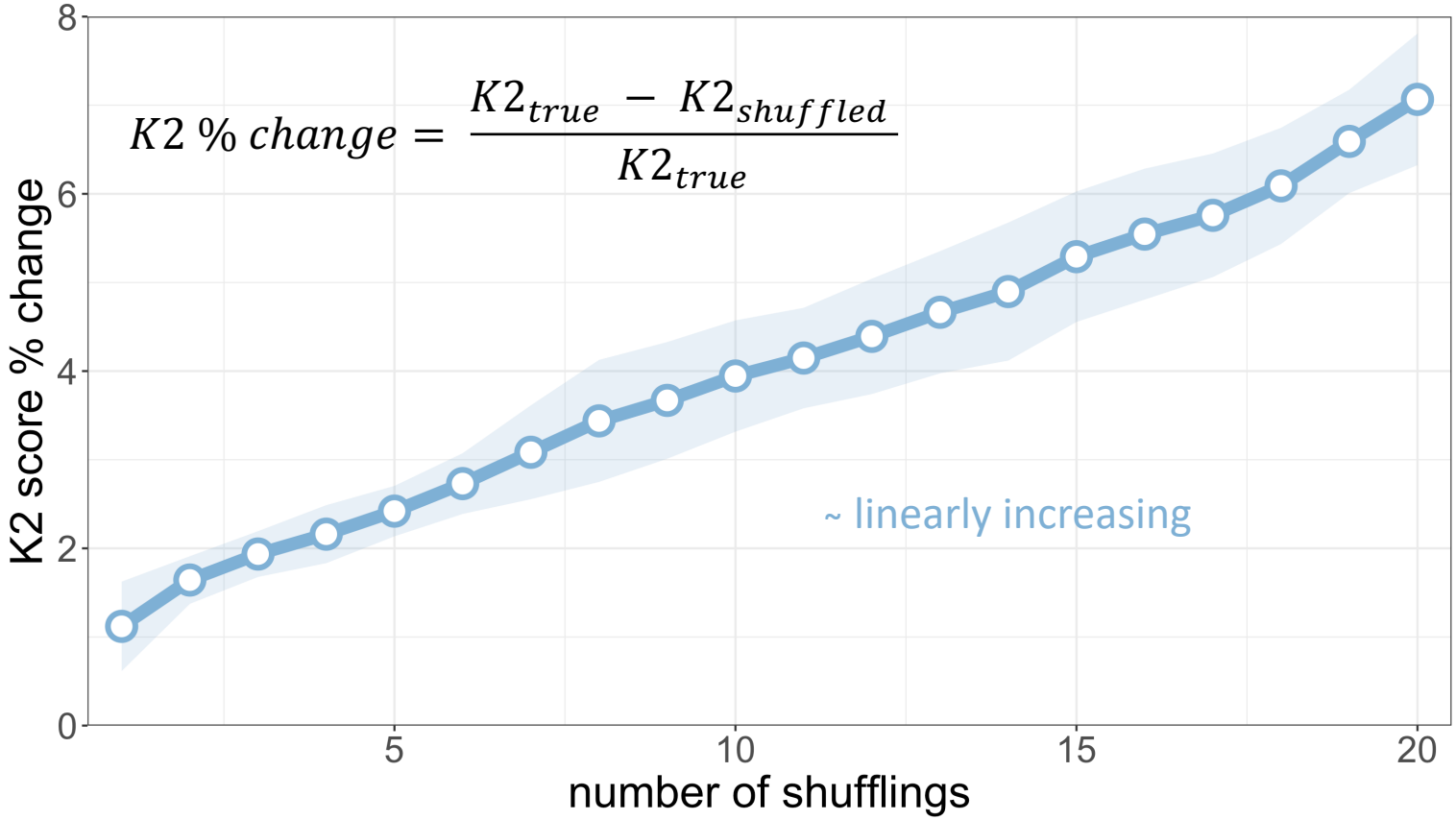
K2 score = *-441633*

NOT ORDERED



K2 score = *-441650*

Influence of feature *ordering* on the network K2 score



Ordering methods

It is critical to provide the *K2 algorithm* the ***correct ordering*** of variables!

We have implemented and tested two **mutual information**-based ordering methods.

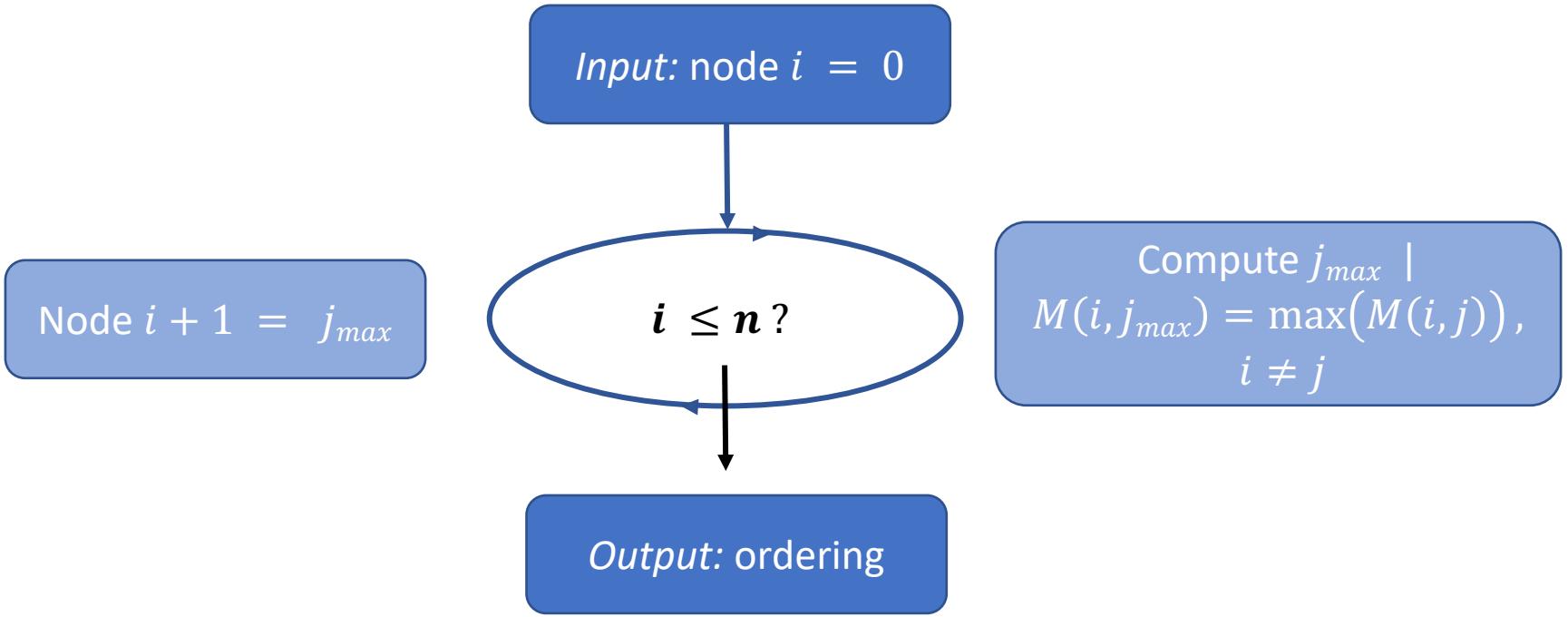
Mutual Information (MI) measures the *degree of dependence* between two random variables:

$$I(X;Y) = H(X) - H(X|Y)$$

where *H(X)* and *H(X|Y)* are the entropy and the conditional entropy respectively.

Mutual Information

The state of a node provides a large amount of information on the state of another node if these two nodes are connected.



SORDER ordering algorithm

The *Sequential ORDER* (SORDER) [1] algorithm takes as input an *UnDirected Graph* (UDG) and selects a suitable ordering of nodes based on their degrees (i.e., the number of adjacent nodes).

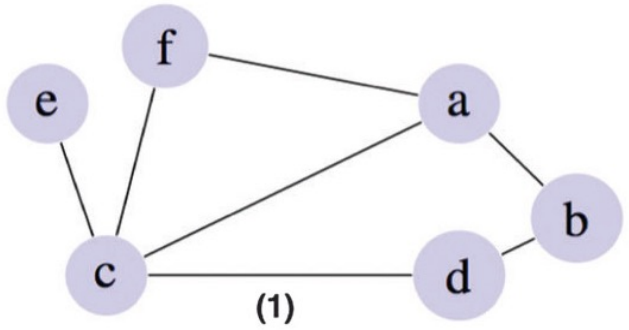
We can build an UDG starting from the dataset linking the variables for which:

$$I(X;Y) \geq \alpha \cdot MMI(X) \text{ or } I(X;Y) \geq \alpha \cdot MMI(Y)$$

Where MMI(X) is the maximum ***MI*** and α is set to 0.9 [2]

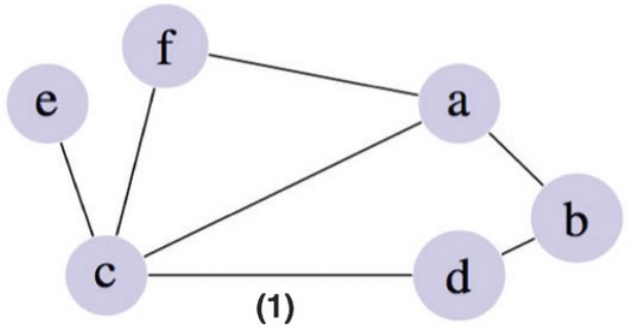
SORDER – an example

Input: UDG



SORDER – an example

Input: UDG

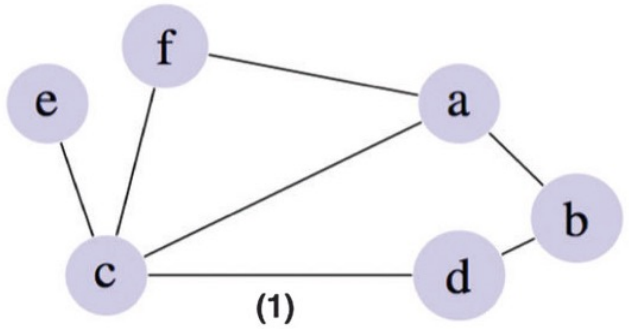


Compute nodes degree

e	c	f	a	d	b
1	4	2	3	2	2

SORDER – an example

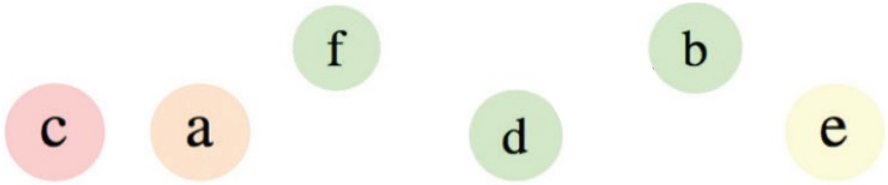
Input: UDG



Compute nodes degree

e	c	f	a	d	b
1	4	2	3	2	2

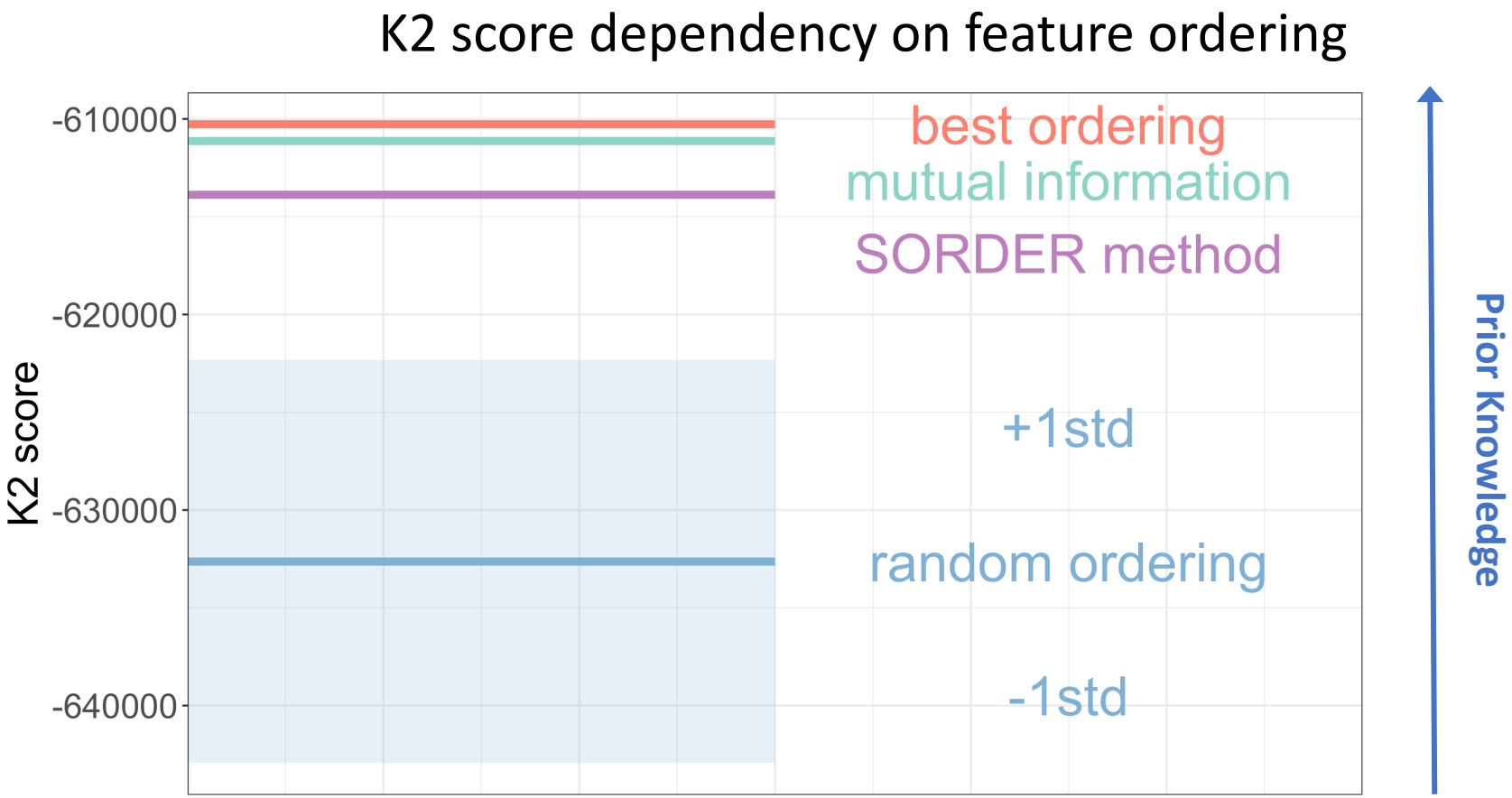
Output: ordering



SORDER – implementation in R

```r
#################### SORDER algorithm ######################
sorder = function(df){
    # compute mi matrix
    mi_mat = mutinformation(df)
    # compute max mi for each node
    mmi = vector()
    for(i in 1:dim(mi_mat)[1]){
        mmi = c(mmi, max(mi_mat[i, -i]))
    }
    # compute UDN edges
    alpha = 0.9
    udn_edges = list()
    for(i in 1:dim(mi_mat)[1]){
        udn_edges_i = vector()
        for(j in 1:dim(mi_mat)[2]){
            if(i==j) next
            if(mi_mat[i,j]>=alpha*mmi[i] || mi_mat[i,j]>=alpha*mmi[j]){
                udn_edges_i = c(udn_edges_i, j)
            }
        }
        udn_edges = append(udn_edges, list(udn_edges_i))
    }
    # compute degrees for each node
    udn_deg = sapply(udn_edges, length)
    udn = data.frame(
        degree = udn_deg,
        node   = colnames(df),
        order  = seq(1, length(udn_deg))
    )
    # create new order for nodes
    udn = udn[order(udn$degree, decreasing=TRUE),]
    s_order = udn$order
    return(s_order)
```

K2 scores in Child Dataset



K2 in `bnstruct`

Implementing the K2 Algorithm inside the `bnstruct` library

The `bnstruct` package

- Provides **objects** and **methods** for *learning the structure and parameters* of the network
- Handles ***missing data*** by performing imputation

  i.e., *guessing the missing value by looking at the data*
- Contains tools to perform *inference* using Bayesian Networks

The `bnstruct` package

- Provides **objects** and **methods** for *learning the structure and parameters* of the network
- Handles ***missing data*** by performing imputation

  i.e., *guessing the missing value by looking at the data*
- Contains tools to perform *inference* using Bayesian Networks

The **main achievement** of the `bnstruct` package is succesfully dealing with ***missing data***!

Implementing the K2 Algorithm inside the `bnstruct` package

- Is *almost* straight-forward
  - the package itself is *well written*
  - the existing code is *clear* and *well commented*

Implementing the K2 Algorithm inside the `bnstruct` package

- Is *almost* straight-forward
  - the package itself is *well written*
  - the existing code is *clear* and *well commented*
- Needs *a few tweaks in the code* to make it fit in the library as a whole
  - though it is not strictly necessary!

Implementing the K2 Algorithm inside the `bnstruct` package

- Is *almost* straight-forward
  - the package itself is *well written*
  - the existing code is *clear* and *well commented*
- Needs *a few tweaks in the code* to make it fit in the library as a whole
  - though it is not strictly necessary!
- Allows access to all the other objects and methods inside the package while using the K2 algorithm to learn the network structure

Conclusions

On learning the *topology* of **Bayesian Networks** through the ***K2* Algorithm**

Achievements
and final considerations

- ***K2* algorithm** implementation in **R**
  - standalone version
  - `bnstruct` version

Achievements
and final considerations

- ***K2* algorithm** implementation in **R**
  - standalone version
  - `bnstruct` version
- Tests on *three* increasing complexity datasets
  - **the initial feature ordering is critical**

Achievements
and final considerations

- ***K2* algorithm** implementation in **R**
  - standalone version
  - `bnstruct` version
- Tests on *three* increasing complexity datasets
  - **the initial feature ordering is critical**
- Exploration of two feature ordering algorithms
  - promising results
  - more complex methods should be explored too [2]

Thank you
for your attention

References

1. Aghdam, R. et al (2015). *CN: A consensus algorithm for inferring gene regulatory networks using the SORDER algorithm and Conditio nal Mutual Information test*. Molecular BioSystems, *11*(3), 942–949. https://doi.org/10.1039/c4mb00413b
2. Xue-Wen C. et al (2008). *Improving bayesian network structure learning with mutual information-based node ordering in the K2 algorithm*. IEEE Transactions on Knowledge and Data Engineering, *20*(5), 628–640. https://doi.org/10.1109/tkde.2007.190732