Bayesian Network and K2 algorithm

Lucia Depaoli, Alessandro Fella
01/07/2021

Università degli Studi di Padova

Introduction

Bayesian Network: probabilistic DAG (Directed Acyclic Graph) model with n nodes connected by edges [1] .

- Nodes $\rightarrow$set of variables in the dataset (continuous or discrete)
- Edges between nodes $\rightarrow$(probabilistic) dependency between nodes

Usually we have a dataset and we want to know the network that have generated the data. This will be the most probable one, between all different combinations of N nodes:

$$max \sum_{i=1}^{N} P(B_{S_i}|D)$$

Issues

- High number of nodes $\rightarrow$high number of possible networks:
  $n = 2$ $\rightarrow$3 possible networks
  $n = 3$ $\rightarrow$25 possible networks
  $n = 5$ $\rightarrow$29000 possible networks
- There could be more than just one structure that maximizes the probability

Basic Model [1]

Theorem:

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

with

$$N_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$$

α_{ijk} $\rightarrow$number of cases in D in which the attribute x_i is instantiated with its k^{th} value, and the parents of x_i in π_i are instantiated with the j^{th} instantiation in ϕ_i.
Observation:

$$P(B_S|D) \propto P(B_S, D)$$

Issue: time complexity.

Assumption

Let's make some assumption in order to reduce the time complexity:

- there is only one structure that maximizes the equation
- specify an order of the n nodes
- equal priors for B_S
- set u for each node
- independent probability for each arc

Now we only need to maximizes the second inner product. We are looking for a set of parents π_i. They have to be consistent with the node ordering.

K2 Algorithm [2]

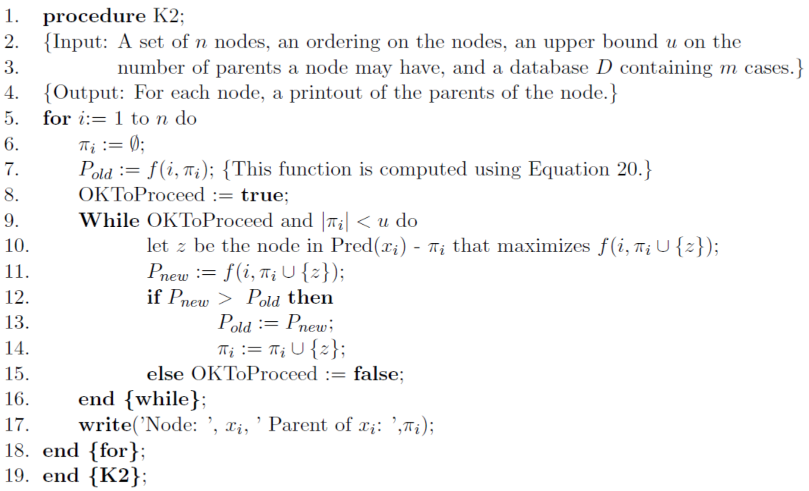
Iterative method:

- $\pi_i = \varnothing$
- add node to π_i if this increases the probability of the structure
- stop adding node if $|\pi_i| = u$ or if it is not possible to increment the probability anymore

Function to compute the probability:

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

K2 algorithm



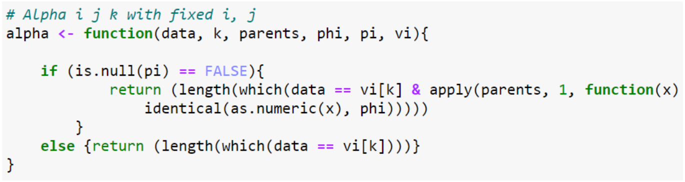
```
procedure K2;
{Input: A set of $n$ nodes, an ordering on the nodes, an upper bound $u$ on the
        number of parents a node may have, and a database $D$ containing $m$ cases.}
{Output: For each node, a printout of the parents of the node.}
for $i$:= 1 to $n$ do
    $\pi_i := \emptyset$;
    $P_{old} := f(i, \pi_i)$; {This function is computed using Equation 20.}
    OKToProceed := true;
    While OKToProceed and $|\pi_i| < u$ do
        let $z$ be the node in Pred($x_i$) - $\pi_i$ that maximizes $f(i, \pi_i \cup \{z\})$;
        $P_{new} := f(i, \pi_i \cup \{z\})$;
        if $P_{new} > P_{old}$ then
            $P_{old} := P_{new}$;
            $\pi_i := \pi_i \cup \{z\}$;
        else OKToProceed := false;
    end {while};
    write('Node: ', $x_i$, ' Parent of $x_i$: ',$\pi_i$);
end {for};
end {K2};
```

Fig: K2 algorithm pseudocode

Our K2 implementation - α_{ijk}

Let's show our *R* implementation for the K2 algorithm.
We have defined 3 function:

- computation of α_{ijk}
- computation of $f(i, \pi_i)$
- loop over the dataset (whole K2 algorithm)



```
# Alpha i j k with fixed i, j
alpha <- function(data, k, parents, phi, pi, vi){

    if (is.null(pi) == FALSE){
            return (length(which(data == vi[k] & apply(parents, 1, function(x)
                identical(as.numeric(x), phi)))))
        }
    else {return (length(which(data == vi[k])))}
}
```

Fig: α_{ijk} function on R

Our K2 implementation - $f(i, \pi_i)$

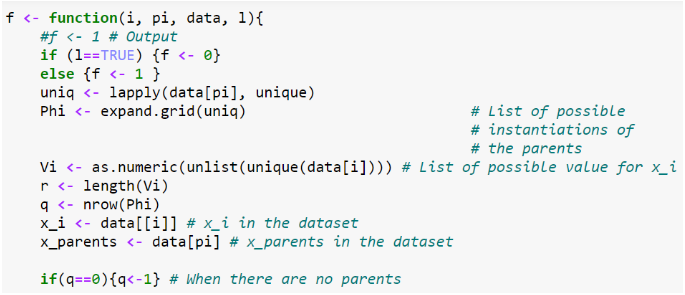
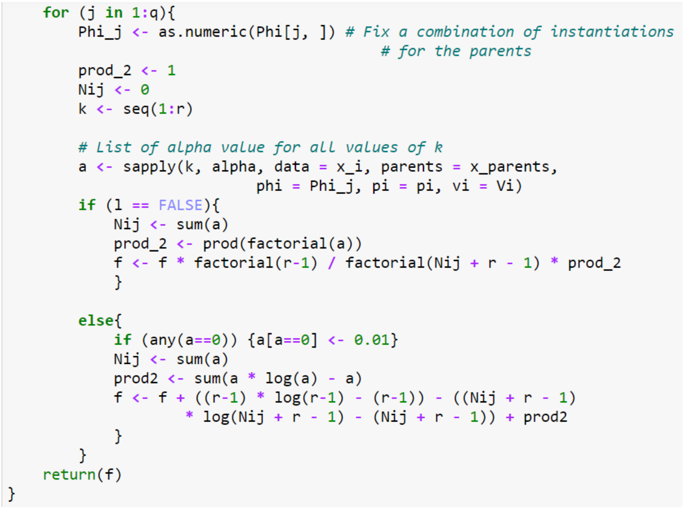

```
f <- function(i, pi, data, l){
    #f <- 1 # Output
    if (l==TRUE) {f <- 0}
    else {f <- 1 }
    uniq <- lapply(data[pi], unique)
    Phi <- expand.grid(uniq)                                # List of possible
                                                            # instantiations of
                                                            # the parents
    Vi <- as.numeric(unlist(unique(data[i]))) # List of possible value for x_i
    r <- length(Vi)
    q <- nrow(Phi)
    x_i <- data[[i]] # x_i in the dataset
    x_parents <- data[pi] # x_parents in the dataset

    if(q==0){q<-1} # When there are no parents
```

Fig: $f(i, \pi_i)$ function on R

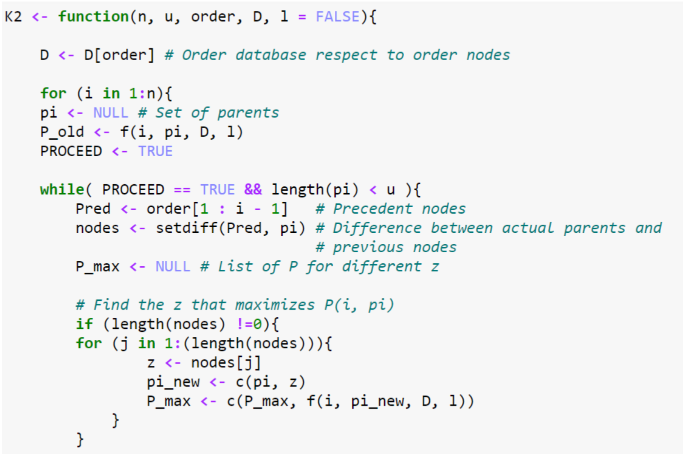
Our K2 implementation - $f(i, \pi_i)$



```
    for (j in 1:q){
        Phi_j <- as.numeric(Phi[j, ]) # Fix a combination of instantiations
                                              # for the parents
        prod_2 <- 1
        Nij <- 0
        k <- seq(1:r)

        # List of alpha value for all values of k
        a <- sapply(k, alpha, data = x_i, parents = x_parents,
                          phi = Phi_j, pi = pi, vi = Vi)
        if (l == FALSE){
            Nij <- sum(a)
            prod_2 <- prod(factorial(a))
            f <- f * factorial(r-1) / factorial(Nij + r - 1) * prod_2
            }

        else{
            if (any(a==0)) {a[a==0] <- 0.01}
            Nij <- sum(a)
            prod2 <- sum(a * log(a) - a)
            f <- f + ((r-1) * log(r-1) - (r-1)) - ((Nij + r - 1)
                    * log(Nij + r - 1) - (Nij + r - 1)) + prod2
            }
        }
    return(f)
}
```

Fig: $f(i, \pi_i)$ function on R

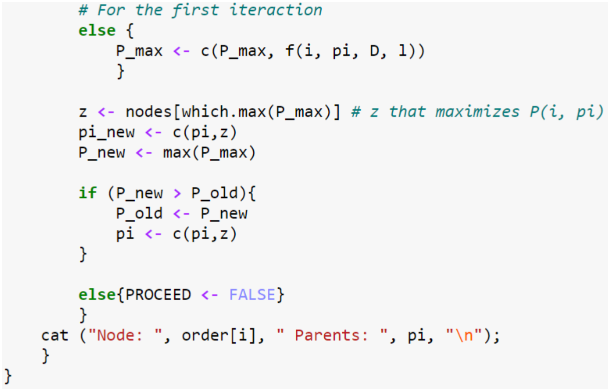
Our K2 implementation - loop over x_i



```r
K2 <- function(n, u, order, D, l = FALSE){

    D <- D[order] # Order database respect to order nodes

    for (i in 1:n){
    pi <- NULL # Set of parents
    P_old <- f(i, pi, D, l)
    PROCEED <- TRUE

    while( PROCEED == TRUE && length(pi) < u ){
        Pred <- order[1 : i - 1]   # Precedent nodes
        nodes <- setdiff(Pred, pi) # Difference between actual parents and
                                   # previous nodes
        P_max <- NULL # List of P for different z

        # Find the z that maximizes P(i, pi)
        if (length(nodes) !=0){
        for (j in 1:(length(nodes))){
                z <- nodes[j]
                pi_new <- c(pi, z)
                P_max <- c(P_max, f(i, pi_new, D, l))
            }
        }
```

Fig: K2 implementation on R

Our K2 implementation - loop over x_i



```
        # For the first iteraction
        else {
            P_max <- c(P_max, f(i, pi, D, 1))
            }

        z <- nodes[which.max(P_max)] # z that maximizes P(i, pi)
        pi_new <- c(pi,z)
        P_new <- max(P_max)

        if (P_new > P_old){
            P_old <- P_new
            pi <- c(pi,z)
        }

        else{PROCEED <- FALSE}
        }
    cat ("Node: ", order[i], " Parents: ", pi, "\n");
    }
}
```

Fig: K2 implementation on R

Our K2 implementation - K2

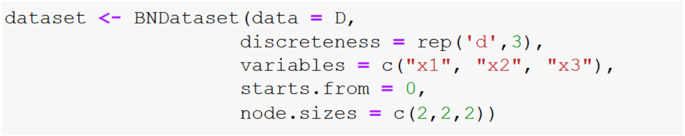
There are 2 different version: the one with the logarithmic version of $f(i, \pi_i)$ and the original one.

- The logarithmic version is faster than the original one
- The logarithmic version is especially useful in the computation of logarithmic with high number of cases

OBSERVATION: the logarithmic version is implemented with a Stirling approximation in order to work on large dataset in R. So, with few cases, it is possible that the output of the two versions are different.

K2 implementation with bnstruct package[3]

The bnstruct package provides objects and methods for learning the structure and parameters of the network given a dataset. Starting from a BNDataset that can be created in the simple following way:



```
dataset <- BNDataset(data = D,
                     discreteness = rep('d',3),
                     variables = c("x1", "x2", "x3"),
                     starts.from = 0,
                     node.sizes = c(2,2,2))
```

Fig: BNDataset from the 1st dataset

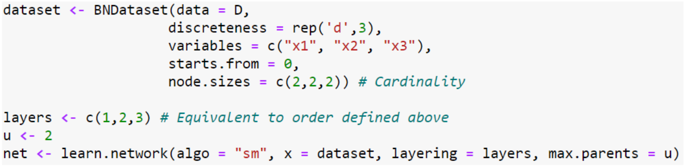
K2 implementation with `bnstruct` package

It is possible to use *five* different algorithms in the learning process.

- `sm` →search-and-score algorithm: performs a complete evaluation of the whole search space
- `mmpc`→constraint-based heuristic approach that is able to find the skeleton of the network
- `hc` →heuristic approach
- `mmhc` →combination of the two previous case
- `sem` →learn a network from a dataset with missing values

K2 implementation with `bnstruct` package

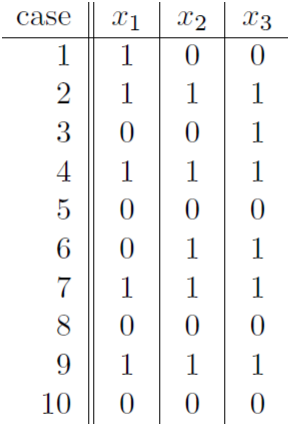
Using the `sm` algorithm it's possible to specify the number of the nodes and their order.
For the starting case (1st dataset) we have the following piece of code showing how to implement the K2 algorithm with `bnstruct` package



```r
dataset <- BNDataset(data = D,
                     discreteness = rep('d',3),
                     variables = c("x1", "x2", "x3"),
                     starts.from = 0,
                     node.sizes = c(2,2,2)) # Cardinality

layers <- c(1,2,3) # Equivalent to order defined above
u <- 2
net <- learn.network(algo = "sm", x = dataset, layering = layers, max.parents = u)
```

Fig: K2 with `bnstruct` package

1^{st} *dataset* [1]

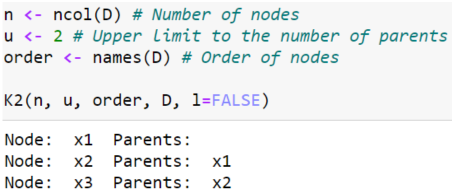


case	x_1	x_2	x_3
1	1	0	0
2	1	1	1
3	0	0	1
4	1	1	1
5	0	0	0
6	0	1	1
7	1	1	1
8	0	0	0
9	1	1	1
10	0	0	0

Fig: 1^{st} *dataset*

Input of K2. $m = 10$ cases. Non-logarithmic version.

- $n = 3$
- $u = (0, 1, 2)$
- $order = x_1, x_2, x_3$



```
n <- ncol(D) # Number of nodes
u <- 2 # Upper limit to the number of parents
order <- names(D) # Order of nodes

K2(n, u, order, D, l=FALSE)
```

```
Node:  x1  Parents:
Node:  x2  Parents:  x1
Node:  x3  Parents:  x2
```

Fig: Output of our K2 implementation

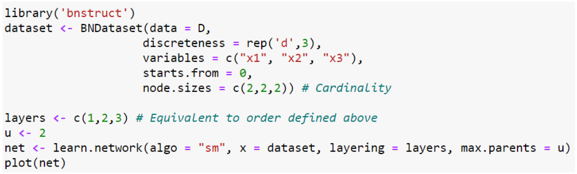
1^{st} *dataset*

This is not the most probabilistic structure for this dataset. If we would have used the Basic Model, we would have found the structure:

$$\begin{tikzcd} x_3 \arrow[r] & x_2 \arrow[r] & x_1 \end{tikzcd}$$

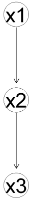
This is because we have fixed an order for the nodes.
Remind that K2 works with approximations.
Using the `bnstruct` we reach the same results.

1^{st} dataset



```
library('bnstruct')
dataset <- BNDataset(data = D,
                     discreteness = rep('d',3),
                     variables = c("x1", "x2", "x3"),
                     starts.from = 0,
                     node.sizes = c(2,2,2)) # Cardinality

layers <- c(1,2,3) # Equivalent to order defined above
u <- 2
net <- learn.network(algo = "sm", x = dataset, layering = layers, max.parents = u)
plot(net)
```

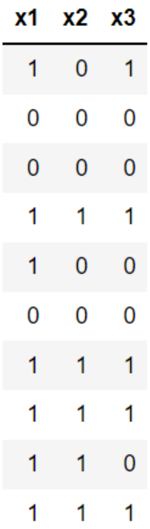
Fig: Implementation with bnstruct



$$\begin{tikzcd} x1 \arrow[d] \\ x2 \arrow[d] \\ x3 \end{tikzcd}$$

Fig: Output of bnstruct

2^{nd} *dataset*



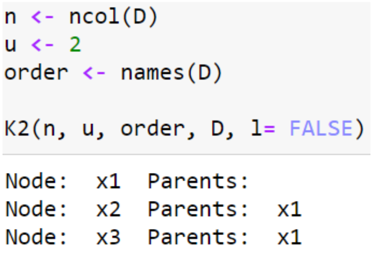
x1	x2	x3
1	0	1
0	0	0
0	0	0
1	1	1
1	0	0
0	0	0
1	1	1
1	1	1
1	1	0
1	1	1

Fig: 2^{nd} *dataset*

From now on we will use dataset generated by ourselves. $m = 10$.

- x_1 is sampled from a binomial distribution with 2 outcomes and $p = 1/2$
- when $x_1 = 1$, then with probability 0.80 $x_2 = 1$, otherwise it will be 0
- when $x_1 = 1$, then with probability 0.60 $x_3 = 1$, otherwise it will be 0.

2^{nd} dataset



```
n <- ncol(D)
u <- 2
order <- names(D)

K2(n, u, order, D, l= FALSE)
```

```
Node:  x1  Parents:
Node:  x2  Parents:  x1
Node:  x3  Parents:  x1
```

Fig: Output of our K2 implementation

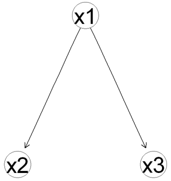
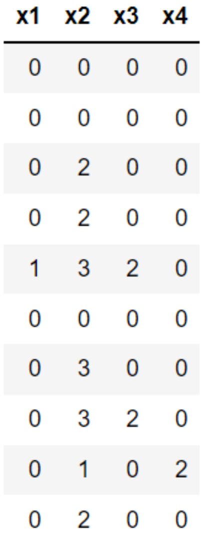


Fig: Output of `bnstruct`

3^{rd} *dataset*



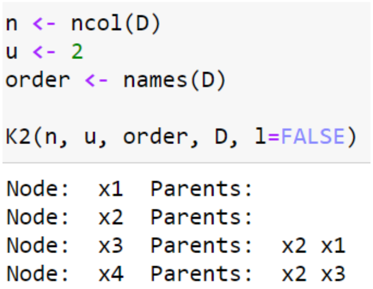
x1	x2	x3	x4
0	0	0	0
0	0	0	0
0	2	0	0
0	2	0	0
1	3	2	0
0	0	0	0
0	3	0	0
0	3	2	0
0	1	0	2
0	2	0	0

Fig: Head of 3^{rd} *dataset*

$m = 50$.

- x_1 is sampled from a binomial distribution with 2 outcomes and $p = 1/5$
- x_2 is sampled using the build-in function `sample(0:3)` with replacement
- when $x_1 = 1$, then with probability 0.80 $x_3 = 1$, otherwise it will be 0
- when $x_2 = 3$, then with probability 0.90 $x_3 = 2$, otherwise it will be 0
- when $x_3 = 1$, with probability 0.70 $x_4 = 1$, otherwise it will be 0
- when $x_2 = 1$, with probability 0.80 $x_4 = 2$, otherwise it will be 0

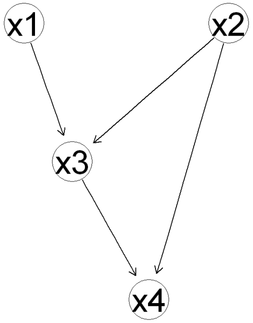
3^{rd} dataset



```
n <- ncol(D)
u <- 2
order <- names(D)

K2(n, u, order, D, l=FALSE)
```

```
Node:   x1   Parents:
Node:   x2   Parents:
Node:   x3   Parents:   x2 x1
Node:   x4   Parents:   x2 x3
```

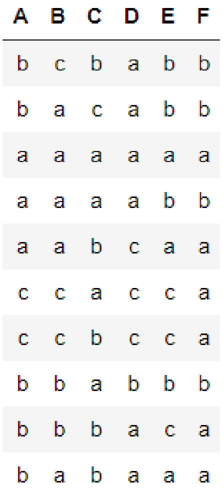
Fig: Output of our K2 implementation



$$\begin{tikzcd}
x1 \arrow[dr] & & x2 \arrow[dl] \arrow[ddl] \\
& x3 \arrow[d] & \\
& x4 &
\end{tikzcd}$$

Fig: Output of `bnstruct`

Learning test dataset

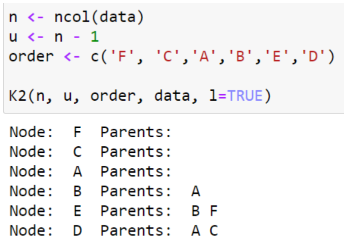


A	B	C	D	E	F
b	c	b	a	b	b
b	a	c	a	b	b
a	a	a	a	a	a
a	a	a	a	b	b
a	a	b	c	a	a
c	c	a	c	c	a
c	c	b	c	c	a
b	b	a	b	b	b
b	b	b	a	c	a
b	a	b	a	a	a

Fig: Head of learning test dataset

We decided to study a more complicated dataset with 6 nodes and 5000 cases. We converted the char values to number from 1 to 3.
We use the log version of the K2.
We have specified an order for the nodes following the true network from `bnstruct` documentation.

Learning test dataset



```
n <- ncol(data)
u <- n - 1
order <- c('F', 'C','A','B','E','D')

K2(n, u, order, data, l=TRUE)
```

```
Node:   F   Parents:
Node:   C   Parents:
Node:   A   Parents:
Node:   B   Parents:   A
Node:   E   Parents:   B F
Node:   D   Parents:   A C
```

Fig: Output of our K2 implementation

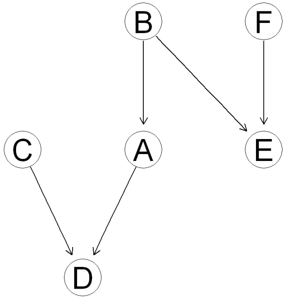


Fig: Output of `bnstruct`

Learning test dataset

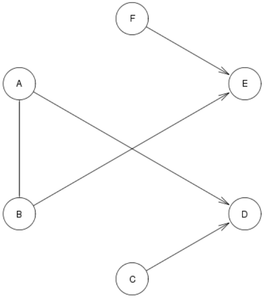
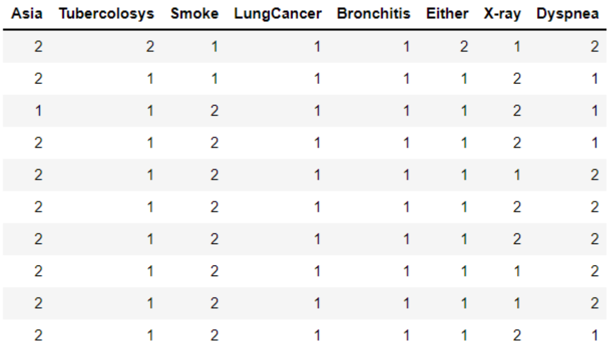


Fig: True Network from `bnlearn` documentation

We can see that the edges between A and B have no direction. So it is not a big deal that our implementation and the output of `bnstruct` implementation gives different results.

Asia dataset

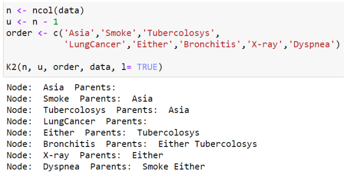
We decided to study a more complicated dataset with 8 nodes and 10000 cases.
This dataset is about lung diseases (tubercolosys, lung cancer or bronchitis) and visits to Asia.



Asia	Tubercolosys	Smoke	LungCancer	Bronchitis	Either	X-ray	Dyspnea
2	2	1	1	1	2	1	2
2	1	1	1	1	1	2	1
1	1	2	1	1	1	2	1
2	1	2	1	1	1	2	1
2	1	2	1	1	1	1	2
2	1	2	1	1	1	2	2
2	1	2	1	1	1	2	2
2	1	2	1	1	1	1	2
2	1	2	1	1	1	1	2
2	1	2	1	1	1	2	1

Fig: Head Asia dataset

Asia dataset



```
n <- ncol(data)
u <- n - 1
order <- c('Asia','Smoke','Tubercolosys',
           'LungCancer','Either','Bronchitis','X-ray','Dyspnea')

K2(n, u, order, data, l= TRUE)
```

```
Node:  Asia  Parents:
Node:  Smoke  Parents:  Asia
Node:  Tubercolosys  Parents:  Asia
Node:  LungCancer  Parents:
Node:  Either  Parents:  Tubercolosys
Node:  Bronchitis  Parents:  Either Tubercolosys
Node:  X-ray  Parents:  Either
Node:  Dyspnea  Parents:  Smoke Either
```

Fig: Output of our K2 implementation

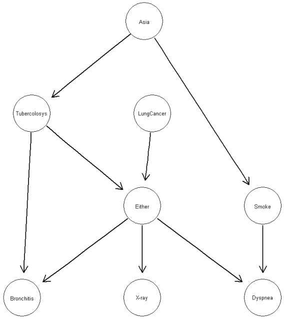


Fig: Output of bnstruct with K2 implementation

Asia dataset

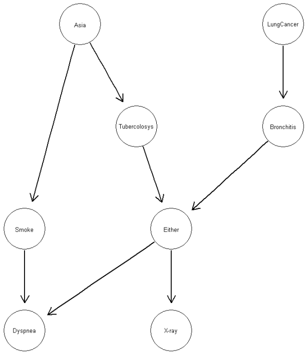


Fig: Output of `bnstruct` without K2 restrictions

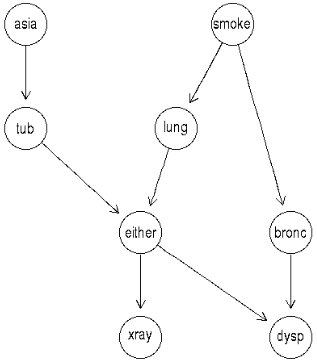


Fig: True structure from `bnlearn` documentation

Time complexity Basic Model

Time complexity:

- complexity for computing equation with N_{ijk} known: $O(m \cdot n \cdot r + t_{B_s})$
- complexity for computing N_{ijk}: $O(m \cdot n^2 \cdot r)$
- **time complexity worst case**: $O(m \cdot n^2 \cdot r + t_{B_s})$

Special cases:

- if we have $u \rightarrow O(m \cdot n \cdot u \cdot r + t_{B_s})$
- if $O(t_{B_s}) = O(u \cdot n \cdot r) \rightarrow O(m \cdot n)$

Time complexity K2 algorithm

The overall complexity is given by

$$O(m + r - 1) + O(m \cdot u \cdot n \cdot r) \cdot O(u) \cdot n$$

where

- $O(m + r - 1)$ because equation shown before contains no factorial greater than (m + r - 1)!
- since each call to f requires $O(m \cdot u \cdot r)$ time so line 10 requires $O(m \cdot u \cdot n \cdot r)$ time
- Each time the *while* statement is entered, it loops $O(u)$ times
- n is due to the *for* statement

Optimizing time complexity

In the worst case, $u = n$, the complexity of K2 algorithm is

$$O(m \cdot n^4 \cdot r)$$

In general it's possible to speed up the computation replacing

- $f(i, \pi_i)$ with $log(f(i, \pi_i))$
- $f(i, \pi_i \bigcup \{z\})$ with $log(f(i, \pi_i \bigcup \{z\}))$

2^{rd} *dataset*

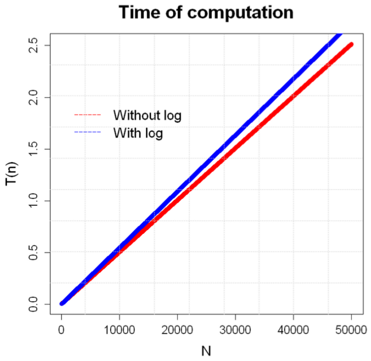


Fig: Time of computation 2^{rd} *dataset*

- *n* fixed
- *m* variable
- we collect time performances using K2 log version and not
- we perform a linear fit of the collected data

Time complexity in the worst case

We build 7 different datasets in which we use different values of *n* and fixed *m*.
Using this approach we can check the time complexity cited in the paper: we can see a $O(n^4)$.

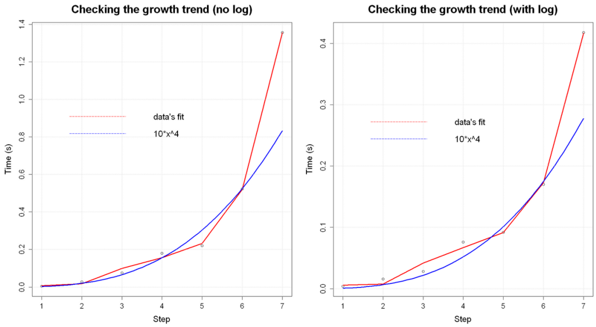


Fig: checking the time complexity in both case

Conclusion

- Our K2 implementation gives expected results
- The K2 implementation using `bnstruct` gives almost the same results as before
- We have been able to reconstruct the structure we have used to make the dataset
- The Asia dataset may not be working correctly due to the complicated structure
- We have correctly verified that the log version of the implemented algorithm is faster then the basic case
- We have verified experimentally the time complexity in the so called "*worst case*" obtaining the $O(n^4)$ trend

Thanks for the attention

bibliography

[1] G. F. Cooper and E. Herskovits, "A bayesian method for the induction of probabilistic networks from data," *Machine learning*, vol. 9, no. 4, pp. 309–347, 1992.

[2] C. Ruiz, "Illustration of the k2 algorithm for learning bayes net structures," *Department of Computer Science, WPI*, 2005.

[3] A. Franzin, F. Sambo, and B. Di Camillo, "Bnstruct: An r package for bayesian network structure learning in the presence of missing data," *Bioinformatics*, vol. 33, no. 8, pp. 1250–1252, 2017.