UNIVERSITÀ DEGLI STUDI DI PADOVA

UNIVERSITÀ DEGLI STUDI DI PADOVA

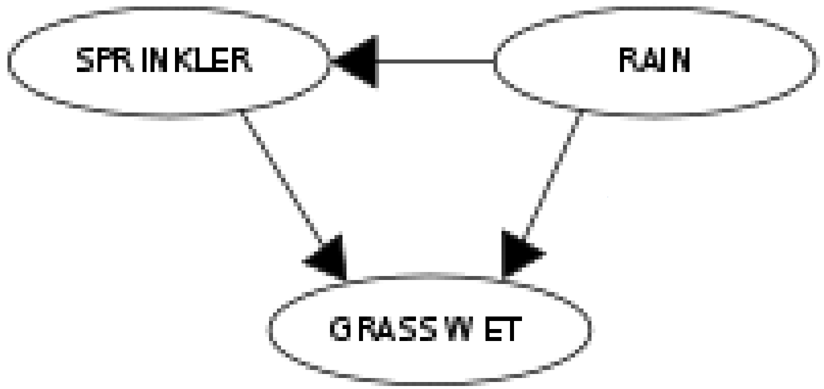
Dipartimento di Fisica e Astronomia “Galileo Galilei ”
Laurea in Physics

Learning the topology of a bayesian network from a database of cases using the K2 algorithm

Giacomo Barzon
Paccagnella Andrea

Bayesian belief-network

	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99

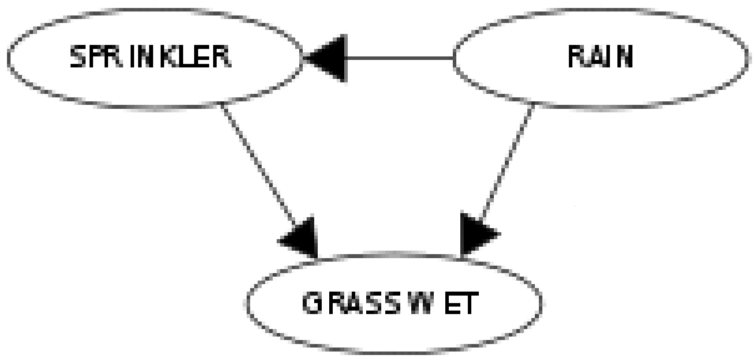


RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Bayesian belief-network

	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99



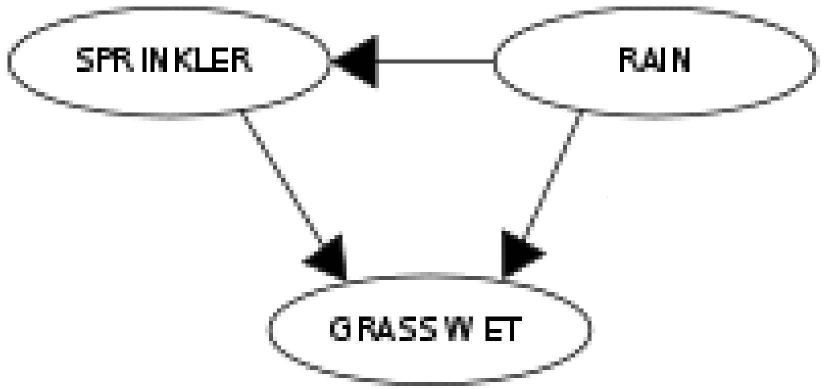
RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

P(G,S,R)=P(G|S,R)*P(S|R)*P(R)

Bayesian belief-network

	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99



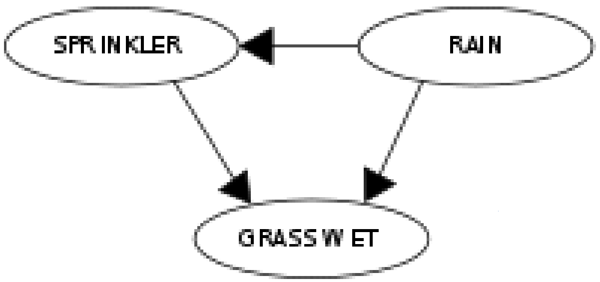
RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

What is the probability that it is raining, given the grass is wet?

Bayesian belief-network

	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99



RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

$$P(R=T\,|\,G=T)=\frac{P(G=T,R=T)}{P(G=T)}=\frac{\sum_{S\in T,F} P(G=T,S,R=T)}{\sum_{S,R\in T,F} P(G=T,S,R)}$$

Bayesian belief-network

	Variable values for each case		
Case	x_1	x_2	x_3
1	*present*	*absent*	*absent*
2	*present*	*present*	*present*
3	*absent*	*absent*	*present*
4	*present*	*present*	*present*
5	*absent*	*absent*	*absent*
6	*absent*	*present*	*present*
7	*present*	*present*	*present*
8	*absent*	*absent*	*absent*
9	*present*	*present*	*present*
10	*absent*	*absent*	*absent*

Bayesian belief-network

Bs

Bp

$P(x_1 = present)$	$= 0.6$	$P(x_1 = absent)$	$= 0.4$
$P(x_2 = present \mid x_1 = present)$	$= 0.8$	$P(x_2 = absent \mid x_1 = present)$	$= 0.2$
$P(x_2 = present \mid x_1 = absent)$	$= 0.3$	$P(x_2 = absent \mid x_1 = absent)$	$= 0.7$
$P(x_3 = present \mid x_2 = present)$	$= 0.9$	$P(x_3 = absent \mid x_2 = present)$	$= 0.1$
$P(x_3 = present \mid x_2 = absent)$	$= 0.15$	$P(x_3 = absent \mid x_2 = absent)$	$= 0.85$

First assumption

$$\frac{P(B_{Si}|D)}{P(B_{Sj}|D)}=\frac{\frac{P(B_{Si},D)}{P(D)}}{\frac{P(B_{Sj},D)}{P(D)}}=\frac{P(B_{Si},D)}{P(B_{Sj},D)}$$

Assumption 1: The database variables, which we denote as Z, are discrete

$$P(B_s,D)=\int_{B_p} P(D|B_s,B_p)\,f(B_p|B_s)\,P(B_s)\,dB_p$$

B_p: a vector whose value denote the conditional probability associated with belief-network structure B_S

f: conditional density function over B_P given B_s

Second assumption

Assumption 2: Cases occur independently, given a belief-network model.

$$P(B_s, D) = \int_{B_p} \left[\prod_{h=1}^{m} P(C_h | B_S, B_P) \right] f(B_p | B_s) P(B_s) \, dB_p$$

m: number of case in D
C_h : is the h-th case in D

Third-Fourth assumption

Assumption 3: There are no cases that have variables with missing values

Assumption 4: The density function f previously seen is uniform

Theorem

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

n: number of node

r_i: number of all possible value of the variable x_i

q_i: number of all possible combination between the value of the parents

α_{ijk}: number of case in D where the node x_i is present with the kth value and the parents of x_i are present with the jth value

N_{ij}: the sum on all k of α_{ijk}

K2 algorithm

$$X_i \longrightarrow X_j \qquad\qquad \cancel{X_j \longrightarrow X_i}$$

$$N^\circ of structure = 2^{n(n-1)/2}$$

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

K2 algorithm

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

π_i: is the parents of X_i

K2 algorithm

DATABASE:

	x1	x2	x3
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

Order of the nodes: 1,2,3

- V_i : list of all possible values of the attribute x_i
- $r_i : |V_i|$

```r
# list of all possible values of the attribute xi
V <- list()

for (i in colnames(data)){
    categories <- sort(unique(data[[i]]))
    V <- c(V, list(categories))
}

# number of possible values for the attribute xi
r <- vector()
for (i in 1:length(V)){
    r <- c(r,length(V[[i]]))
}
```

K2 algorithm

- ϕ: list of all possible instantiations of the parents of x_i in database D
- q_i: $|\phi_i|$

```
# cartesian product for two or more lists
cartesian <- function(...) {
    axb <- expand.grid(...)
    axb <- axb[complete.cases(axb),]

    # order from first to last column
    for (j in ncol(axb):1){
        axb <- axb[order(axb[j]),]
    }
    return(as.matrix(axb))
}
```

K2 algorithm

```r
# ph i: list of all possible instantiations of the parents of xi in the database
ph <- function(parents){
    if (length(parents) == 0){
        ph <- 0
    }
    else if (length(parents) == 1){
        ph <- t(t(V[[parents]]))   # in order to get a column vector
    }
    else{
        temp <- list()
        for (i in 1:length(parents)){
            temp <- c(temp, list(V[[parents[i]]]))
        }
        ph <- cartesian(temp)
    }
    return (ph)
}

# q: total number of possible instantiations of the parents of xi in the database
q <- function(parents){
    ifelse(length(parents)>0, nrow(ph(parents)), 0)
}
```

K2 algorithm

- α_{ijk}: number of cases in D in which the attribute x_i is instantiated with its kth value, and the parents of x_i in π_i are instantiated with the jth instantiation in ϕ_i
- $N_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$: the number of instances in the database in which the parents of x_i in π_i are instantiated with the jth instantiation in ϕ_i

```r
# alpha i,j,k: number of cases in which:
# - xi attribute is instantiated with its k value
# - parents of xi are instantiated with its j value
alpha <- function(pi,i,j,k){
    phi <- ph(pi)

    # xi with its k value
    sub <- data[data[,i] == V[[i]][k], ]

    # parents of xi with its j value
    if(j>0){
        for (l in 1:length(pi)){
            sub <- sub[sub[,pi[l]] == phi[j,l], ]
        }
    }

    return(nrow(sub))
}
```

```r
# N i,j: number of cases in which:
# - xi attribute is instantiated with its k value
# - parents of xi are instantiated with its j value
Nij <- function(pi,i,j){
    Nij <- 0
    for (k in 1:r[i]){
        Nij <- Nij + alpha(pi,i,j,k)
    }

    return(Nij)
}
```

K2 algorithm

- Pred(x_i): a function that returns the set of nodes that precede x_i in the node ordering

```
# Pred(xi): returns the set of nodes that precede xi
pred <- function(i){
    return (head(order, match(i,order)-1))
}
```

K2 algorithm

$f(i, \pi_i)$: **probability of the database D given that the parents of x_i are π_i**

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

BUT in order to save run-time, we compute the logarithm of f(i,π_i)

- $\log[f(i, \pi_i)] = \sum_{j=1}^{q_i} \log[(r_i - 1)!] - \log[(N_{ij} + r_i - 1)!] + \sum_{k=1}^{r_i} \log[\alpha_{ijk}!]$
  $= \sum_{j=1}^{q_i} logfact[r_i - 1] - logfact[N_{ij} + r_i - 1] + \sum_{k=1}^{r_i} logfact[\alpha_{ijk}]$

K2 algorithm

```r
# log(f(i,pi)) computation:
logf <- function(i, pi){
    qi <- q(pi)

    sum <- 0

    # If list of parents is not empty
    if(qi>0){
        for (j in 1:qi){
            for (k in 1:r[i]){
                aijk <- alpha(pi,i,j,k)
                sum <- sum + lfactorial(aijk)
            }
            nij <- Nij(pi,i,j)
            sum <- sum + lfactorial(r[i]-1) - lfactorial(nij + r[i]-1)
        }
    }
    # If list of parents is empty
    else{
        for (k in 1:r[i]){
            ai0k <- alpha(pi,i,0,k)
            sum <- sum + lfactorial(ai0k)
        }
        ni0 <- Nij(pi,i,0)
        sum <- sum + lfactorial(r[i]-1) - lfactorial(ni0 + r[i]-1)
    }

    return(sum)
}
```

K2 algorithm

Network conditional probability: $\theta_{ijk} = p(x_i = v_{ik} | \pi_i = \omega_{ij})$

$$\vartheta_{ijk} = \frac{\alpha_{ijk}+1}{N_{ij}+r_i}$$

Probability that node x_i has value v_{ik} (k from 1 to r_i), given that the parents of x_i, represented by π_i, are instatiated as ω_{ij}

```
# theta i,j,k computation:
theta <- function(pi,i){

    if (length(pi)==0){
        theta <- matrix(0, 1, (r[i]))
        for(k in 1:r[i]){
            theta[1,k]<-(alpha(pi,i,0,k)+1)/(Nij(pi,i,0)+r[i])
        }
    }

    else{
        theta <- matrix(0, (r[i]),q(pi))
    for (j in 1:q(pi)){
        for(k in 1:r[i]){
            theta[k,j]<-(alpha(pi,i,j,k)+1)/(Nij(pi,i,j)+r[i])
        }
    }
}
    return(signif(theta,2))
}
```

K2 algorithm

$$Var_{ijk} = \frac{(\alpha_{ijk}+1)(N_{ij}+r_i-\alpha_{ijk}-1)}{(N_{ij}+r_i)^2(N_{ij}+r_i+1)}$$

```
# variance of theta i,j,k
var <- function(pi,i){

    if(length(pi)==0){
        var <- matrix(0, 1,(r[i]))
        for(k in 1:r[i]){
            var[1,k] <- ( (alpha(pi,i,0,k)+1) * (Nij(pi,i,0) + r[i] - alpha(pi,i,0,k)-1)) /
                        ( (Nij(pi,i,0) + r[i]+1) * (Nij(pi,i,0) + r[i])^2 )
        }
    }

    else{
        var <- matrix(0, (r[i]),q(pi))
    for (j in 1:q(pi)){
        for(k in 1:r[i]){
            var[k,j] <- ( (alpha(pi,i,j,k)+1) * (Nij(pi,i,j) + r[i] - alpha(pi,i,j,k)-1)) /
                        ( (Nij(pi,i,j) + r[i]+1) * (Nij(pi,i,j) + r[i])^2 )
        }
    }
        }
    return(signif(var,1))
}
```

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) - π_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: ',π_i);
end **{for}**;
end **{K2}**;

** We compute log(f(i,π_i)) instead of f(i,π_i) in order to save computation time and reduce complexity
(since log is a monotonic function)

K2 algorithm

for i:= 1 to n do
  $\pi_i := \emptyset$;
  $P_{old} := f(i, \pi_i)$; {This function is computed using Equation 20.}
  OKToProceed := **true**;

```r
# K2 algorithm:
# - return: adjacency matrix between all nodes (as a data frame)
K2 <- function(verbose=TRUE){
    cat("RUNNING K2 ALGORITHM\n\n")
    start.time <- Sys.time()

    adjMat <- matrix(0, n,n) # adjacency matrix initialization

    for (i in 1:n){
        pi <- vector()

        Pold <- logf(i,pi)
        OkToProceed <- TRUE
```

K2 algorithm

While OKToProceed and $|\pi_i| < u$ do
let z be the node in $\text{Pred}(x_i)$ - π_i that maximizes $f(i, \pi_i \cup \{z\})$;
$P_{new} := f(i, \pi_i \cup \{z\})$;

```r
while (OkToProceed & length(pi)<u){
    # z: node in Pred(xi) - PIi
    z <- pred(i)
    z <- z[! z %in% pi]

    # if Pred(xi) is empty -> iteration ends with pi = empty
    if (length(z)==0){
        OkToProceed = FALSE
        break
    }

    # find the node z that maximizes f(i, pi U z)
    Pnew <- -Inf
    newParent <- 0

    for(k in 1:length(z)){
        tempP <- logf(i, sort(c(pi,z[k])))
        if(tempP > Pnew){
            Pnew <- tempP
            newParent <- z[k]
        }
    }
```

K2 algorithm

$$
\begin{aligned}
&\quad\textbf{if } P_{new} > P_{old} \textbf{ then} \\
&\qquad\quad P_{old} := P_{new}; \\
&\qquad\quad \pi_i := \pi_i \cup \{z\}; \\
&\quad\textbf{else } \text{OKToProceed} := \textbf{false}; \\
&\textbf{end } \{\textbf{while}\}; \\
&\textbf{write}(\text{'Node: ', } x_i, \text{ ' Parent of } x_i\text{: '}, \pi_i);
\end{aligned}
$$

```r
        # if Pnew > Pold, add z to parents list
        # otherwise, iteration ends
        if(Pnew > Pold){
            Pold <- Pnew
            pi <- sort(c(pi, newParent))
            adjMat[newParent, i] <- 1
        }
        else{
            OkToProceed = FALSE
        }
    } # end while
    cat("Node:",i,", Parent of node",i,": ",pi,"\n")

    if (verbose==TRUE){
        cat(theta.print(pi,i),"\n")
    }
}
```

K2 algorithm

end {for};
end {K2};

```
    # add node names to adjacency matrix
    adjMat <- data.frame(adjMat)
    names(adjMat) <- names
    row.names(adjMat) <- names

    # print adjacency matrix
    if (verbose==TRUE){
        cat("\nAdjacency matrix:\n\n")
        print(adjMat)
    }

    end.time <- Sys.time()
    total.time <- end.time - start.time
    cat("\nTotal computation time:",total.time,"mins\n")

    return (adjMat)
}
```

K2 results

```
RUNNING K2 ALGORITHM

Node: 1 , Parent of node 1 :
P( x1 = 0 ) = 0.5 ± 0.02
P( x1 = 1 ) = 0.5 ± 0.02

Node: 2 , Parent of node 2 :  1
P( x2 = 0 | x1 = 0 ) = 0.71 ± 0.03
P( x2 = 1 | x1 = 0 ) = 0.29 ± 0.03
P( x2 = 0 | x1 = 1 ) = 0.29 ± 0.03
P( x2 = 1 | x1 = 1 ) = 0.71 ± 0.03

Node: 3 , Parent of node 3 :  2
P( x3 = 0 | x2 = 0 ) = 0.71 ± 0.03
P( x3 = 1 | x2 = 0 ) = 0.29 ± 0.03
P( x3 = 0 | x2 = 1 ) = 0.14 ± 0.02
P( x3 = 1 | x2 = 1 ) = 0.86 ± 0.02


Adjacency matrix:

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

Total computation time: 0.1709189 mins
```

bnstruct package

bnstruct: an R package for Bayesian Network Structure Learning with missing data

Francesco Sambo, Alberto Franzin

December 12, 2016

```r
# learn BN object
bnNet <- learn.network(dataset.from.data, algo="MMHC")
```


```
Bayesian Network: BNDataset

num.nodes 3

variables
x1 x2 x3
discreteness
TRUE TRUE TRUE
node.sizes
2 2 2
Adjacency matrix:
   x1 x2 x3
x1  0  0  0
x2  1  0  0
x3  0  1  0

Conditional probability tables:$x1
   x1
x2          1         2
  1 0.7727273 0.2272727
  2 0.2272727 0.7727273

$x2
   x2
x3          1          2
  1 0.9444444 0.05555556
  2 0.1923077 0.80769231

$x3
x3
        1         2
0.4090909 0.5909091
```

K2 implementation in *bnstruct*

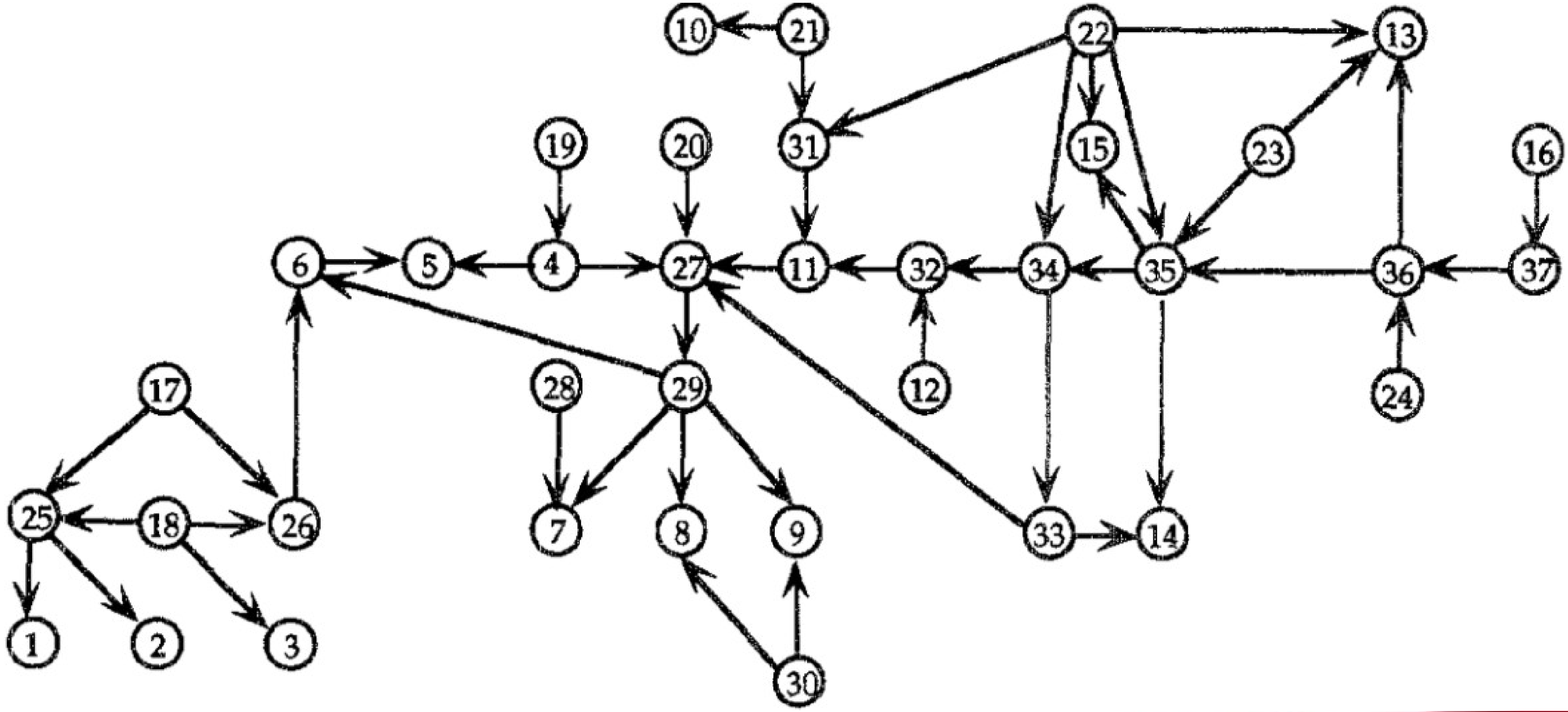
```
# BNDataset object creation
dataMat <- as.matrix(data)
dataset.from.data <- BNDataset(data = dataMat, discreteness = c('d','d','d'),
                               variables = names, node.sizes = r, starts.from=0)

# BN object creation
net <- BN(dataset.from.data)
# add the adjacency matrix to the BN object
dag(net) <- as.matrix(adjMatrix)
```

ALARM network

ALARM ("A Logical Alarm Reduction Mechanism"):
Bayesian network designed to provide an alarm message system for patient monitoring

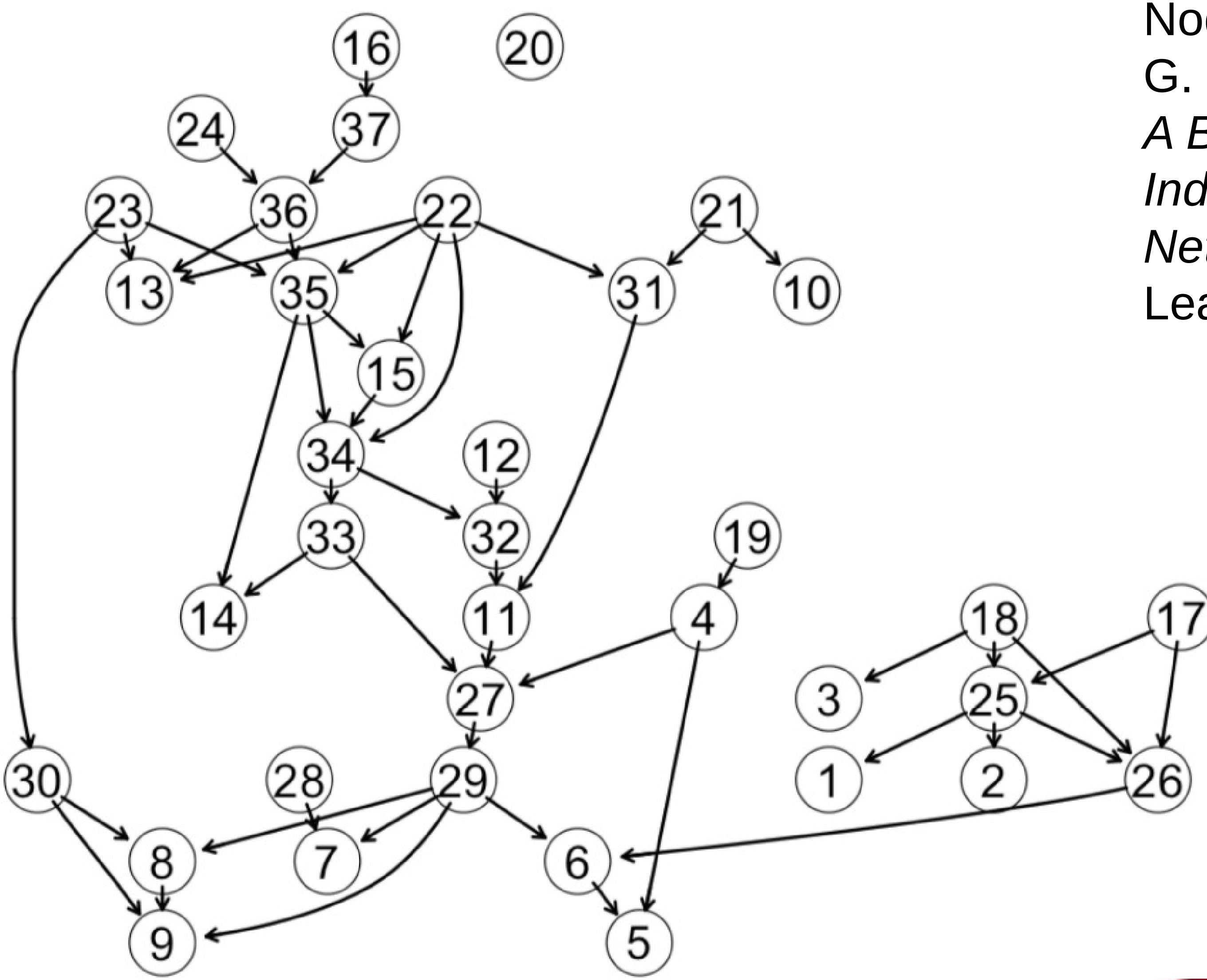
8 diagnosis, 16 findings, 13 intermediate variables

e.g.:
- 27: increased release of adrenaline
- 29: increased heart rate
- 8: EKG measuring increased heart rate



K2 with ALARM

Dataset: 10.000 cases generated from ALARM network
http://www.openmarkov.org/learning/



Node order specify in:
G. F. Cooper and E. Herskovits, *A Bayesian Method for the Induction of Probabilistic Networks from Data*, Machine Learning 9, (1992) 309

Errors:
- missing arc from node 20 to node 27
- adding arc from node 23 to node 30
- adding arc from node 15 to node 34

K2

≠

MMHC algorithm

Difference of time between the two database

Total computation time 3 column: 0.2389789 mins

Total computation time Alarm: 17.11669 mins