00-tutorial

September 23, 2019

[1]:
```
%matplotlib inline
from pylab import *
import matplotlib.pyplot as plt
import os
```

1 Creating your first Bayesian Network with pyAgrum

(This example is based on an OpenBayes [closed] website tutorial)

A Bayesian network (BN) is composed of random variables (nodes) and their conditional dependencies (arcs) which, together, form a directed acyclic graph (DAG). A conditional probability table (CPT) is associated with each node. It contains the conditional probability distribution of the node given its parents in the DAG:

Such a BN allows to manipulate the joint probability $P(C,S,R,W)$ using this decomposition :

$P(C,S,R,W) = \prod_X P(X|Parents_X) = P(C)\cdot P(S|C)\cdot P(R|C)\cdot P(W|S,R)$

Imagine you want to create your first Bayesian network, say for example the 'Water Sprinkler' network. This is an easy example. All the nodes are Boolean (only 2 possible values). You can proceed as follows.

1.1 Import the pyAgrum package

[2]:
```
import pyAgrum as gum
```

1.2 Create the network topology

1.2.1 Create the BN

The next line creates an empty BN network with a 'name' property.

[3]:
```
bn=gum.BayesNet('WaterSprinkler')
print(bn)
```

```
BN{nodes: 0, arcs: 0, domainSize: 1, dim: 0}
```

1.2.2 Create the variables

pyAgrum(aGrUM) provides 3 types of variables :

LabelizedVariable

RangeVariable

DiscretizedVariable

In this tutorial, we will use LabelizedVariable, which is a variable whose domain is a finite set of labels. The next line will create a variable named 'c', with 2 values and described as 'cloudy?', and it will add it to the BN. The value returned is the id of the node in the graphical structure (the DAG). pyAgrum actually distinguishes the random variable (here the labelizedVariable) from its node in the DAG: the latter is identified through a numeric id. Of course, pyAgrum provides functions to get the id of a node given the corresponding variable and conversely.

[4]:
```
c=bn.add(gum.LabelizedVariable('c','cloudy ?',2))
print(c)
```

```
0
```

You can go on adding nodes in the network this way. Let us use python to compact a little bit the code:

[5]:
```
s, r, w = [ bn.add(name, 2) for name in "srw" ]
#bn.add(name, 2) === bn.add(gum.LabelizedVariable(name, name, 2))
print (s,r,w)
print (bn)
```

```
1 2 3
BN{nodes: 4, arcs: 0, domainSize: 16, dim: 8}
```

1.2.3 Create the arcs

Now we have to connect nodes, i.e., to add arcs linking the nodes. Remember that c and s are ids for nodes:

[6]:
```
bn.addArc(c,s)
```

Once again, python can help us :

[7]:
```
for link in [(c,r),(s,w),(r,w)]:
    bn.addArc(*link)
print(bn)
```

```
BN{nodes: 4, arcs: 4, domainSize: 16, dim: 18}
```

pyAgrum provides tools to display bn in more user-frendly fashions. Notably, pyAgrum.lib is a set of tools written in pyAgrum to help using aGrUM in python. pyAgrum.lib.notebook adds dedicated functions for iPython notebook.

[8]:
```
import pyAgrum.lib.notebook as gnb
bn
```

[8]:
```
(gum::BayesNet<double>@0x56391a865670) BN{nodes: 4, arcs: 4, domainSize: 16,
dim: 18}
```

1.2.4 Create the probability tables

Once the network topology is constructed, we must initialize the conditional probability tables (CPT) distributions. Each CPT is considered as a Potential object in pyAgrum. There are several ways to fill such an object.

To get the CPT of a variable, use the cpt method of your BayesNet instance with the variable's id as parameter.

Now we are ready to fill in the parameters of each node in our network. There are several ways to add these parameters.

Low-level way

[9]:
```
bn.cpt(c).fillWith([0.5,0.5])
```

[9]:
```
(gum::Potential<double>@0x56391a70bf30) <c:0> :: 0.5 /<c:1> :: 0.5
```

Most of the methods using a node id will also work with name of the random variable.

[10]:
```
bn.cpt("c").fillWith([0.4,0.6])
```

[10]:
```
(gum::Potential<double>@0x56391a70bf30) <c:0> :: 0.4 /<c:1> :: 0.6
```

Using the order of variables

[11]:
```
bn.cpt(s).var_names
```

[11]:
```
['c', 's']
```

[12]:
```
bn.cpt(s)[:]=[ [0.5,0.5],[0.9,0.1]]
```

Then $P(S|C=0)=[0.5,0.5]$ and $P(S|C=1)=[0.9,0.1]$.

[13]:
```
print(bn.cpt(s)[1])
```

```
[0.9 0.1]
```

The same process can be performed in several steps:

[14]:
```
bn.cpt(s)[0,:]=0.5 # equivalent to [0.5,0.5]
bn.cpt(s)[1,:]=[0.9,0.1]
```

[15]:
```
bn.cpt(w).var_names
```

[15]:
```
['r', 's', 'w']
```

[16]:
```
bn.cpt(w)[0,0,:] = [1, 0] # r=0,s=0
bn.cpt(w)[0,1,:] = [0.1, 0.9] # r=0,s=1
bn.cpt(w)[1,0,:] = [0.1, 0.9] # r=1,s=0
bn.cpt(w)[1,1,:] = [0.01, 0.99] # r=1,s=1
```

Using a dictionnary This is probably the most convenient way:

[17]:
```
bn.cpt(w)[{'r': 0, 's': 0}] = [1, 0]
bn.cpt(w)[{'r': 0, 's': 1}] = [0.1, 0.9]
bn.cpt(w)[{'r': 1, 's': 0}] = [0.1, 0.9]
bn.cpt(w)[{'r': 1, 's': 1}] = [0.01, 0.99]
bn.cpt(w)
```

[17]:
```
(gum::Potential<double>@0x56391a869f20) <w:0|s:0|r:0> :: 1 /<w:1|s:0|r:0> :: 0
/<w:0|s:1|r:0> :: 0.1 /<w:1|s:1|r:0> :: 0.9 /<w:0|s:0|r:1> :: 0.1 /<w:1|s:0|r:1>
:: 0.9 /<w:0|s:1|r:1> :: 0.01 /<w:1|s:1|r:1> :: 0.99
```

The use of dictionaries is a feature borrowed from OpenBayes. It facilitates the use and avoid common errors that happen when introducing data into the wrong places.

[18]:
```
bn.cpt(r)[{'c':0}]=[0.8,0.2]
bn.cpt(r)[{'c':1}]=[0.2,0.8]
```

1.3 Input/output

Now our BN is complete. It can be saved in different format :

[19]:
```
print(gum.availableBNExts())
```

```
bif|dsl|net|bifxml|o3prm|uai
```

We can save a BN using BIF format

[20]:
```
# 'out' folder has to exist
gum.saveBN(bn,os.path.join("out","WaterSprinkler.bif"))
```

[21]:
```
with open(os.path.join("out","WaterSprinkler.bif"),"r") as out:
    print(out.read())
```

```
network "WaterSprinkler" {
// written by aGrUM 0.16.2.9
}

variable c {
   type discrete[2] {0, 1};
}
```

```
variable s {
   type discrete[2] {0, 1};
}

variable r {
   type discrete[2] {0, 1};
}

variable w {
   type discrete[2] {0, 1};
}

probability (c) {
   default 0.4 0.6;
}
probability (s | c) {
   (0) 0.5 0.5;
   (1) 0.9 0.1;
}
probability (r | c) {
   (0) 0.8 0.2;
   (1) 0.2 0.8;
}
probability (w | s, r) {
   (0, 0) 1 0;
   (1, 0) 0.1 0.9;
   (0, 1) 0.1 0.9;
   (1, 1) 0.01 0.99;
}
```

[22]:
```
bn2=gum.loadBN(os.path.join("out","WaterSprinkler.bif"))
```

We can also save and load it in other formats

[23]:
```
gum.saveBN(bn,os.path.join("out","WaterSprinkler.net"))
with open(os.path.join("out","WaterSprinkler.net"),"r") as out:
    print(out.read())
bn3=gum.loadBN(os.path.join("out","WaterSprinkler.net"))
```

```
net {
  name = WaterSprinkler;
  software = "aGrUM 0.16.2.9";
  node_size = (50 50);
}
```

```
node c {
   states = (0 1 );
   label = "c";
   ID = "c";
}

node s {
   states = (0 1 );
   label = "s";
   ID = "s";
}

node r {
   states = (0 1 );
   label = "r";
   ID = "r";
}

node w {
   states = (0 1 );
   label = "w";
   ID = "w";
}

potential (c) {
   data = (  0.4 0.6);
}

potential ( s | c   ) {
   data =
   ((   0.5   0.5)   % c=0
   (   0.9   0.1));   % c=1
}

potential ( r | c   ) {
   data =
   ((   0.8   0.2)   % c=0
   (   0.2   0.8));   % c=1
}

potential ( w | s   r   ) {
   data =
   (((   1   0)   % s=0   r=0
   (   0.1   0.9))   % s=1   r=0
   ((   0.1   0.9)   % s=0   r=1
   (   0.01   0.99)));   % s=1   r=1
}
```

2 Inference in Bayesian Networks

We have to choose an inference engine to perform calculations for us. Two inference engines are currently available in pyAgrum:

LazyPropagation: an exact inference method that transforms the Bayesian network into a hypergraph called a join tree or a junction tree. This tree is constructed in order to optimize inference computations.

Gibbs : an approximate inference engine using the Gibbs sampling algorithm to generate a sequence of samples from the joint probability distribution.

[24]:
```
ie=gum.LazyPropagation(bn)
```

2.1 Inference without evidence

[25]:
```
ie.makeInference()
print (ie.posterior(w))
```

```
<w:0> :: 0.33328 /<w:1> :: 0.66672
```

In our BN, $P(W) = [0.3529, \quad 0.6471]$

With notebooks, it can be viewed as an HTML table

[26]:
```
ie.posterior(w)
```

[26]:
```
(gum::Potential<double>@0x56391a1a5a00) <w:0> :: 0.33328 /<w:1> :: 0.66672
```

2.2 Inference with evidence

Suppose now that you know that the sprinkler is on and that it is not cloudy, and you wonder what Is the probability of the grass being wet, i.e., you are interested in distribution $P(W|S = 1, C = 0)$. The new knowledge you have (sprinkler is on and it is not cloudy) is called evidence. Evidence is entered using a dictionary. When you know precisely the value taken by a random variable, the evidence is called a hard evidence. This is the case, for instance, when I know for sure that the sprinkler is on. In this case, the knowledge is entered in the dictionary as 'variable name':label

[27]:
```
ie.setEvidence({'s':0, 'c': 0})
ie.makeInference()
ie.posterior(w)
```

[27]:
```
(gum::Potential<double>@0x56391a5e51e0) <w:0> :: 0.82 /<w:1> :: 0.18
```

When you have incomplete knowledge about the value of a random variable, this is called a soft evidence. In this case, this evidence is entered as the belief you have over the possible values that

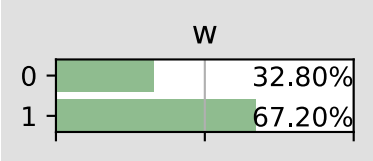
the random variable can take, in other words, as P(evidence|true value of the variable). Imagine for instance that you think that if the sprinkler is off, you have only 50% chances of knowing it, but if it is on, you are sure to know it. Then, your belief about the state of the sprinkler is [0.5, 1] and you should enter this knowledge as shown below. Of course, hard evidence are special cases of soft evidence in which the beliefs over all the values of the random variable but one are equal to 0.

[28]:
```
ie.setEvidence({'s': [0.5, 1], 'c': [1, 0]})
ie.makeInference()
ie.posterior(w) # using gnb's feature
```

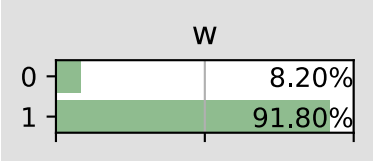
[28]:
```
(gum::Potential<double>@0x563919d10ae0) <w:0> :: 0.328 /<w:1> :: 0.672
```

the pyAgrum.lib.notebook utility proposes certain functions to graphically show distributions.

[29]:
```
%matplotlib inline
gnb.showProba(ie.posterior(w))
```

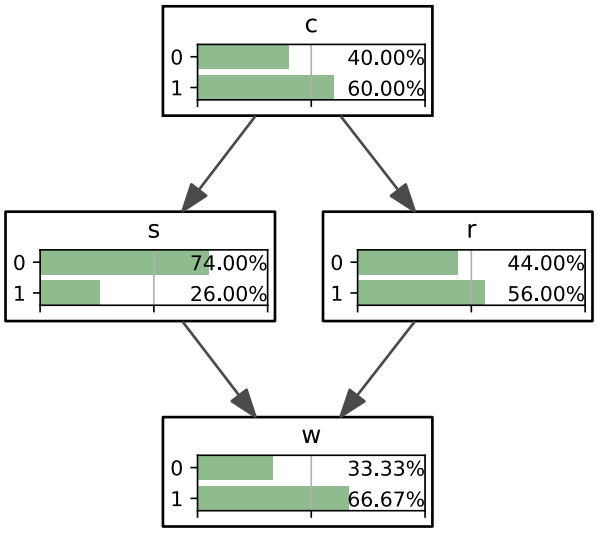


[30]:
```
gnb.showPosterior(bn,{'s':1,'c':0},'w')
```



2.3 inference in the whole Bayes net

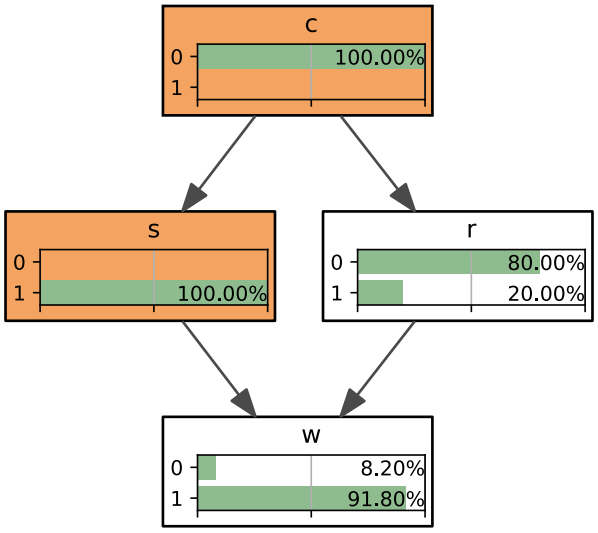
[31]:
```
gnb.showInference(bn,evs={})
```



Inference in 0.55ms

2.3.1 inference with evidence

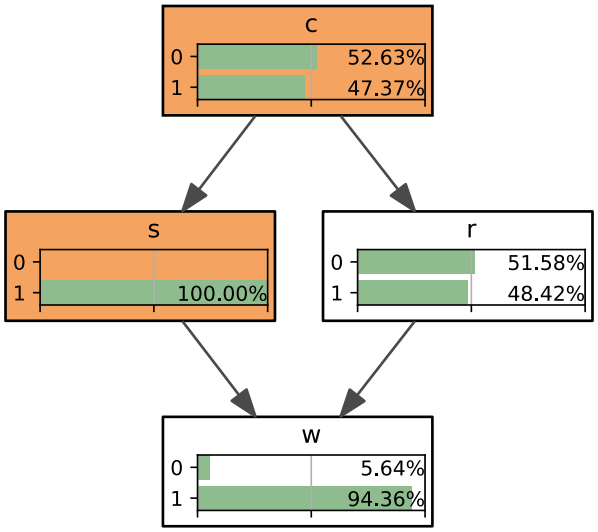
[32]:
```
gnb.showInference(bn,evs={'s':1,'c':0})
```



Inference in 0.70ms

2.3.2 inference with soft and hard evidence

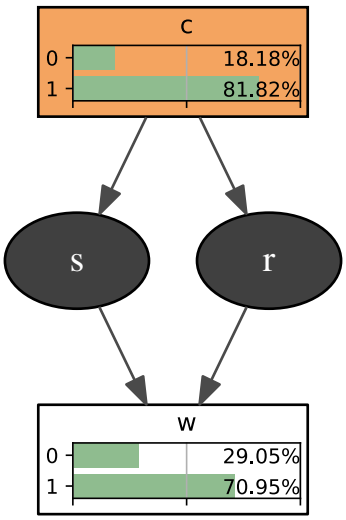
[33]:
```
gnb.showInference(bn,evs={'s':1,'c':[0.3,0.9]})
```



Inference in 1.14ms

2.3.3 inference with partial targets

[34]:
```
gnb.showInference(bn,evs={'c':[0.3,0.9]},targets={'c','w'})
```



Inference in 0.71ms

[ ]:

01-Probabilités

September 23, 2019

[1]:
```
import pyAgrum as gum
import pyAgrum.lib.notebook as gnb

from IPython.display import display, Math, Latex
```

1 Daltonisme bayésien

Environ 8% des hommes et 0.5% des femmes sont, à des degrés divers, daltoniens.

1.1 Question 1

Calculer le pourcentage de femmes parmi les daltoniens (en ajoutant une hypothèse a priori que vous préciserez)

[2]:
```
genre=gum.LabelizedVariable("G","Gender",0).addLabel("H").addLabel("F")
dalton=gum.LabelizedVariable("D","Daltonism",0).addLabel("O").addLabel("N")

display(Math(r"$P(D|G)$"))
pDsiG=gum.Potential().add(dalton).add(genre).fillWith([8/100,92/100,0.5/100,99.
↪5/100])
gnb.showPotential(pDsiG)

pG=gum.Potential().add(genre).fillWith([0.48,0.52])
display(Math(r"$P(G)$"))
gnb.showPotential(pG)

pDetG=pDsiG*pG
display(Math(r"$P(G,D)=P(D|G)\cdot P(G)$"))
gnb.showPotential(pDetG)

pD=pDetG.margSumOut(["G"])
display(Math(r"$P(D)=\sum_G P(G,D)$"))
gnb.showPotential(pD)

display(Math(r"$P(G|D)=\frac{P(G,D)}{p(D)}"))
pGsiD=pDetG/pD
gnb.showPotential(pGsiD)
```

```
print("Il y a {:4.2f}% de femmes parmi les daltoniens".format(100*pGsiD[{"D":
 ↪"O","G":"F"}]))
```

$P(D|G)$

```
<IPython.core.display.HTML object>
```

$P(G)$

```
<IPython.core.display.HTML object>
```

$P(G,D) = P(D|G) \cdot P(G)$

```
<IPython.core.display.HTML object>
```

$P(D) = \sum_G P(G,D)$

```
<IPython.core.display.HTML object>
```

$P(G|D) = \frac{P(G,D)}{p(D)}$

```
<IPython.core.display.HTML object>
```

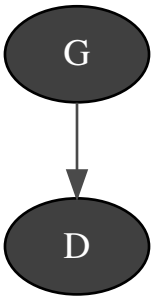
```
Il y a 6.34% de femmes parmi les daltoniens
```

1.2 Question 2

Construire le réseau bayésien qui représente ce problème et vérifier cette valeur par propagation.

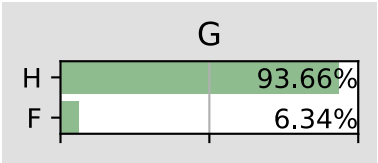
[3]:
```
bn=gum.BayesNet()

g=bn.add(genre)
d=bn.add(dalton)

bn.addArc(g,d)

# ou alors en une seule ligne
bn=gum.fastBN("G{H|F}->D{O|N}")
```

[4]:
```
bn.cpt("G").fillWith(pG[:])

bn.cpt("D")[{'G':'H'}]=[0.08,0.92]
bn.cpt("D")[{'G':'F'}]=[0.005,0.995]

gnb.showBN(bn)
```



Il suffit de faire une inférence de l'information $D = 0$ (Daltonien=oui)

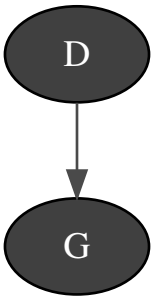
[5]:
```
gnb.showPosterior(bn,target='G',evs={'D':0})
```



2 Question 2 (bis)

Une autre façon de faire le calcul : renverser l'arc pour obtenir un BN qui contient $P(D)$

[6]:
```
bn.reverseArc(g,d)
gnb.showBN(bn)
```



[7]:
```
gnb.showPotential(bn.cpt(g))
```

```
<IPython.core.display.HTML object>
```

On retrouve bien $P(G = F|D = O) = 0.0634$

[ ]:

02-CPTdeterministe

September 23, 2019

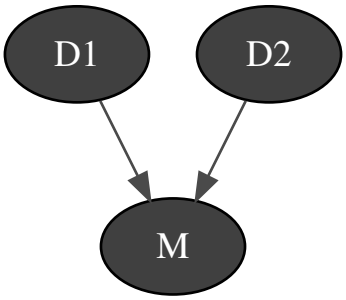
[1]:
```
import pyAgrum as gum
import pyAgrum.lib.notebook as gnb
from IPython.display import display, Math, Latex
```

1 Exercice 2

On lance 2 dés (équilibrés). Déterminer, grâce à un réseau bayésien la loi de probabilité du maximum des chiffres indiqués par les 2 dés.

PS- on peut représenter des dépendances fonctionnelles entre variables par des CPTs ne contenant que des probabilités 0 ou 1

[2]:
```
bn=gum.BayesNet()

d1=bn.add(gum.RangeVariable("D1","Dé 1",1,6))
d2=bn.add(gum.RangeVariable("D2","Dé 2",1,6))

m=bn.addMAX(gum.RangeVariable("M","Max(D1,D2)",1,6))

bn.addArc(d1,m)
bn.addArc(d2,m)

bn.cpt(d1).fillWith([1]*6).normalize()
bn.cpt(d2).fillWith([1]*6).normalize()
```



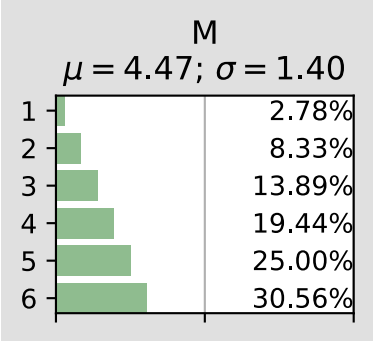
[8]:
```
# de manière équivalent (mais plus rapide)
bn2=gum.fastBN("D1[1,6]->M[1,6]<-D2[1,6]")
bn2.cpt("D1").fillWith([1]*6).normalize()
bn2.cpt("D2").fillWith([1]*6).normalize()
bn2.cpt("M").fillWithFunction("max(D1,D2)")
```

[9]:
```
gnb.sideBySide(bn,bn2)

gnb.sideBySide(bn.cpt("M"),bn2.cpt("M"))
```

```
<IPython.core.display.HTML object>


<IPython.core.display.HTML object>
```

Calculer la distribution de la somme : a priori,

[23]:
```
#histogramme
gnb.showPosterior(bn,target='M',evs={})

# liste
print(gum.getPosterior(bn,target='M',evs={}).tolist())

# potentiel
gum.getPosterior(bn,target='M',evs={})
```

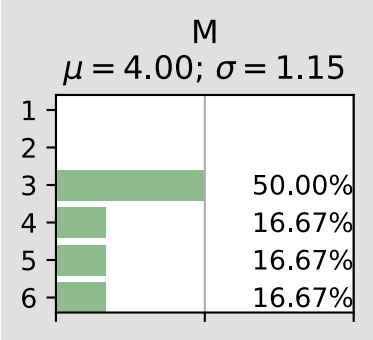



```
[0.027777777777777776, 0.08333333333333333, 0.1388888888888889,
0.19444444444444442, 0.24999999999999997, 0.3055555555555555]
```

[23]:
```
(gum::Potential<double>@0x7fa3d9510da0) <M:1> :: 0.0277778 /<M:2> :: 0.0833333
/<M:3> :: 0.138889 /<M:4> :: 0.194444 /<M:5> :: 0.25 /<M:6> :: 0.305556
```

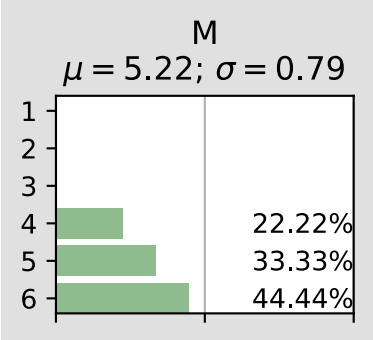
en supposant que le tirage du dé 1 vaut 3,

[24]:
```
gnb.showPosterior(bn,target='M',evs={"D1":2}) # 3 est d'index 2
```



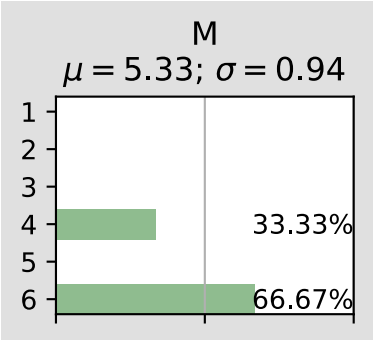
puis qu'il est plus grand que 3,

[25]:
```
gnb.showPosterior(bn,target='M',evs={"D1":[0,0,0,1,1,1]})
```



puis en ajoutant que le maximum est pair

[26]:
```
gnb.showPosterior(bn,target='M',evs={"D1":[0,0,0,1,1,1],"M":[0,1,0,1,0,1]})
```



03-Modélisation1

September 23, 2019

[1]:
```
import pyAgrum as gum
import pyAgrum.lib.notebook as gnb

from IPython.display import display, Math, Latex
```

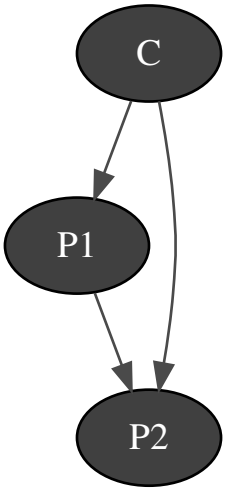
1 Exercice 3

Trois coffres contiennent respectivement une pièce d'or et une pièce d'argent, 2 pièces d'or et 2 pièces d'argent. On choisit une pièce dans un des trois coffres. C'est une pièce d'or. Quelle est la probabilité pour que la seconde pièce soit également en or ? (Problème posé par le mathématicient Joseph Bertrand)

1- Modéliser le problème comme un BN

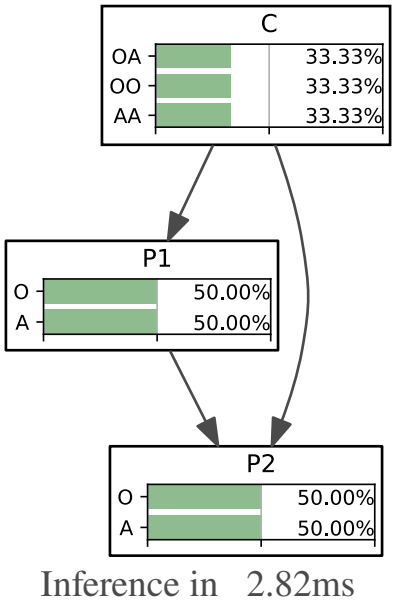
[3]:
```
#bn=gum.BayesNet()

#c=bn.add(gum.LabelizedVariable("C","Coffres",0).addLabel("OA").addLabel("OO").
 ↪addLabel("AA"))
#p1=bn.add(gum.LabelizedVariable("P1","Pièce 1",0).addLabel("O").addLabel("A"))
#p2=bn.add(gum.LabelizedVariable("P2","Pièce 2",0).addLabel("O").addLabel("A"))

#bn.addArc(c,p1)
#bn.addArc(c,p2)
#bn.addArc(p1,p2)
bn=gum.fastBN("C{OA|OO|AA}->P1{O|A}->P2{O|A}<-C")
bn.cpt("C").fillWith([1]*3).normalize()

bn.cpt("P1")[{'C':'OA'}]=[0.5,0.5]
bn.cpt("P1")[{'C':'OO'}]=[1,0]
bn.cpt("P1")[{'C':'AA'}]=[0,1]

bn.cpt("P2")[{'C':'OA','P1':'O'}]=[0,1]
bn.cpt("P2")[{'C':'OO','P1':'O'}]=[1,0]
bn.cpt("P2")[{'C':'AA','P1':'O'}]=[0.5,0.5]
bn.cpt("P2")[{'C':'OA','P1':'A'}]=[1,0]
bn.cpt("P2")[{'C':'OO','P1':'A'}]=[0.5,0.5]
bn.cpt("P2")[{'C':'AA','P1':'A'}]=[0,1]
```

```
gnb.showBN(bn)
```

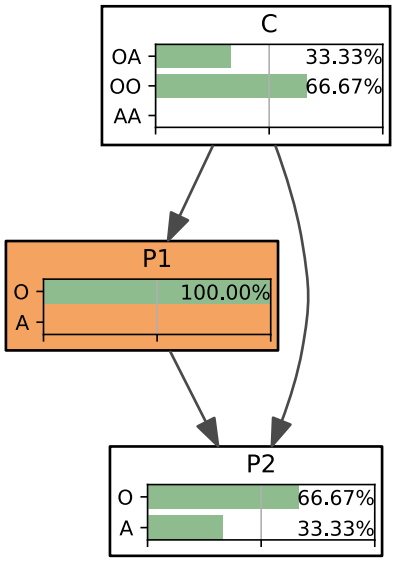


2- résoudre par inférence dans le BN

[4]:
```
gnb.showInference(bn)
```



[5]:
```
gnb.showInference(bn,evs={'P1':'O'})
```



Inference in 1.45ms

[ ]:

04-Modélisation2

September 23, 2019

[1]:
```
import pyAgrum as gum
import pyAgrum.lib.notebook as gnb

from IPython.display import display, Math, Latex
```

1 Hector le robot jongleur

from Roland Donat

Hector, le robot jongleur, lâche souvent les balles avec lesquelles il jongle quand sa batterie est faible. D'après les expériences précédentes, il a été déterminé que la probabilité qu'il lâche une balle quand sa batterie est faible est de 0.9. D'autre part, quand sa batterie n'est pas faible, la probabilité qu'il lâche une balle est seulement de 0.01. Quand la batterie a été rechargée il y a peu de temps, il y a seulement 5% de chances que la batterie soit faible contre 50% si la batterie n'a pas été rechargée récemment.

Un premier système de vision $O1$ (peu fiable) observe le robot et nous prévient lorsqu'il croit que Hector a lâché une balle. Un autre système $O2$ (indépendant du premier) agit de la même façon. Le but de cet exercice est de modéliser les résultats des observateurs en fonction de ce que fait Hector et de son état.

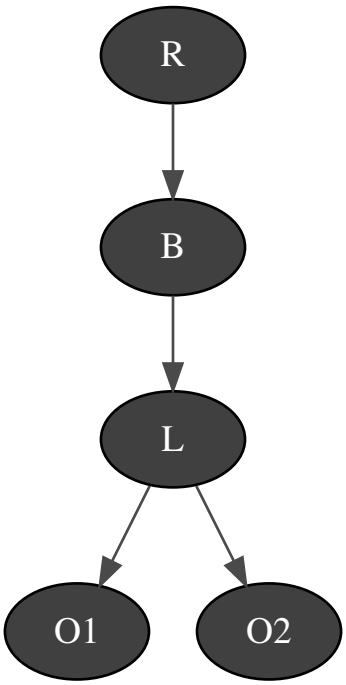
1.1 Structure du réseau bayésien

Construire un BN pour ce problème en considérant :

- Quelles variables (booléenne) de modélisation ?
- Quelle structure ?
- Quelle est la forme de la distribution de probabilité jointe pour ce BN
- $O1$ et $O2$ sont ils indépendantes ? A quelle condition deviennent-elles indépéndantes ?

[2]:
```
bn=gum.BayesNet()

R=bn.add(gum.LabelizedVariable("R","Rechargement",0).addLabel("yalongtemps").
 ↪addLabel("toutjuste"))
B=bn.add(gum.LabelizedVariable("B","Batterie d'Hector",0).addLabel("vide").
 ↪addLabel("pleine"))
L=bn.add(gum.LabelizedVariable("L","Hector lâche la balle",0).addLabel("lâche").
 ↪addLabel("OK"))
```

```
O1=bn.add(gum.LabelizedVariable("O1","Observateur 1",0).addLabel("vu").
 ↪addLabel("pasvu"))
O2=bn.add(gum.LabelizedVariable("O2","Observateur 2",0).addLabel("vu").
 ↪addLabel("pasvu"))

bn.addArc(R,B)
bn.addArc(B,L)
bn.addArc(L,O1)
bn.addArc(L,O2)

gnb.showBN(bn)
```

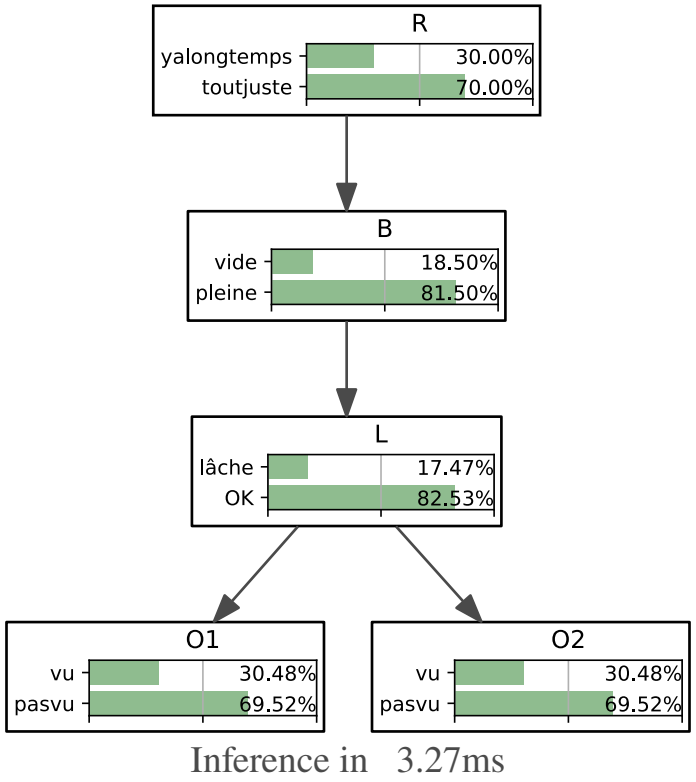


1.2 Paramètres du réseau bayésien

Proposer les CPTs pour le BN. Quelle données vous manque-t-il pour avoir l'ensemble du modèle bien renseigné ?

[3]:
```
bn.cpt(R).fillWith([0.3,0.7]) #on ne les a pas dans l'énoncé

bn.cpt(B)[{"R":"yalongtemps"}]=[0.5,0.5]
bn.cpt(B)[{"R":"toutjuste"}]=[0.05,0.95]

bn.cpt(L)[{"B":"vide"}]=[0.9,0.1]
bn.cpt(L)[{"B":"pleine"}]=[0.01,0.99]
```

```
bn.cpt(O1).fillWith([0.8,0.2,0.2,0.8])  #on ne les a pas dans l'énoncé
bn.cpt(O2).fillWith([0.8,0.2,0.2,0.8]) #on ne les a pas dans l'énoncé

gnb.showInference(bn)
```

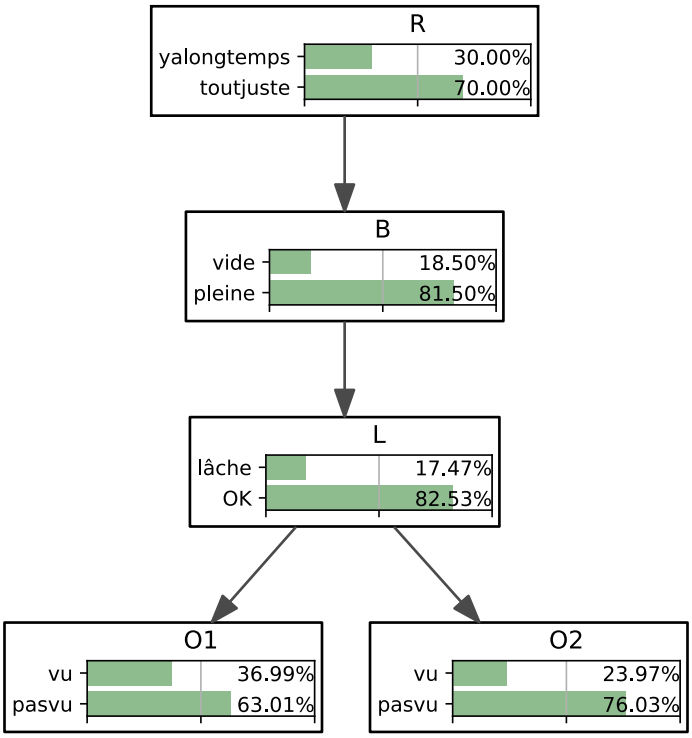


1.3 inférence dans le réseau bayésien

On suppose maintenant que la fiabilité des observateurs $O1$ et $O2$ est de respectivement 70% et 90%.

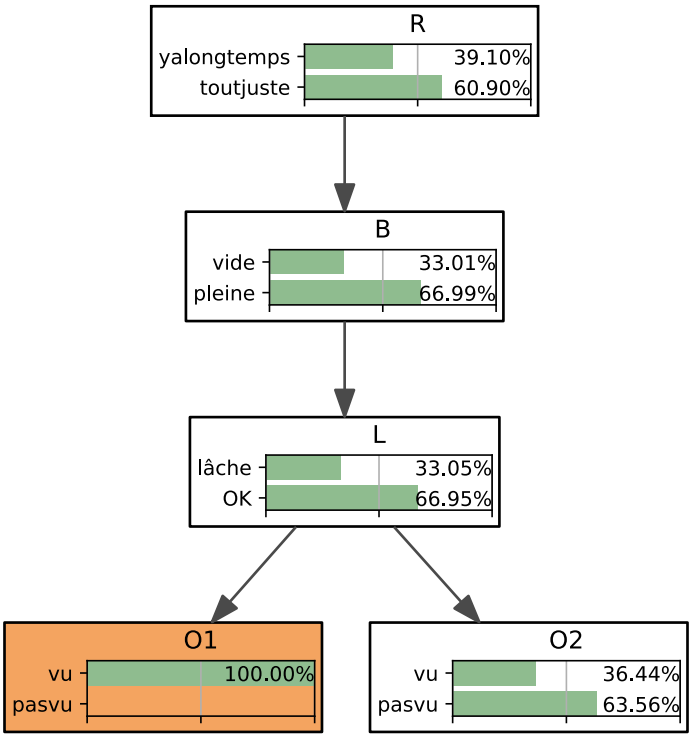
- Calculer la probabilité a priori de chaque nœud
- $O1$ observe que Hector a lâché la balle. Quelle est la probabilité que la batterie soit faible ? Comparer cette valeur avec sa probabilité à priori.
- On ajoute que $O2$ n'a rien vu. Quelle est la nouvelle probabilité que la batterie soit faible ?

[4]:
```
bn.cpt(O1).fillWith([0.7,0.3,0.3,0.7])  #on ne les a pas dans l'énoncé
bn.cpt(O2).fillWith([0.9,0.1,0.1,0.9]) #on ne les a pas dans l'énoncé

gnb.showInference(bn)
```



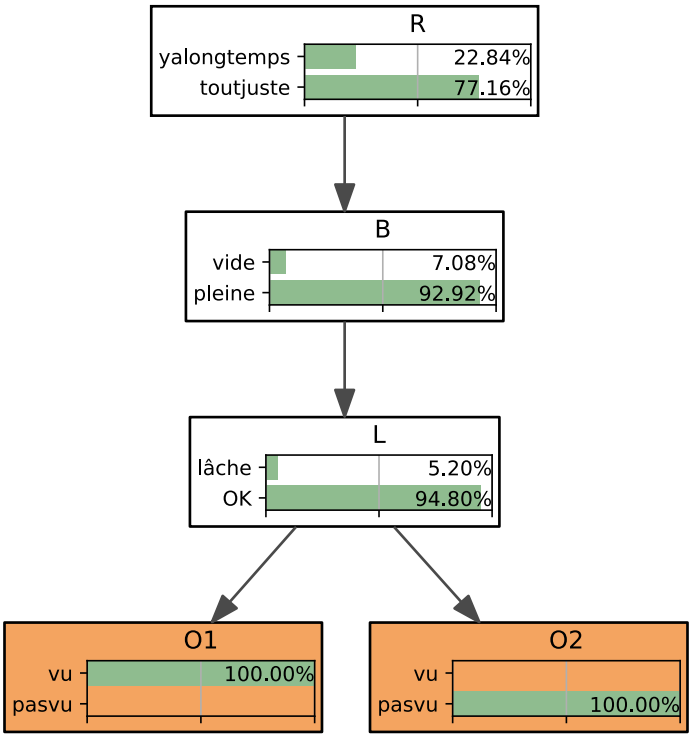
Inference in 0.79ms

[5]:
```
gnb.showInference(bn,evs={'O1':'vu'})
```



Inference in 0.63ms

[6]:
```
gnb.showInference(bn,evs={'O1':'vu','O2':'pasvu'})
```



Inference in 1.10ms

[ ]:

05-Modélisation3

September 23, 2019

[1]:
```
import pyAgrum as gum
import pyAgrum.lib.notebook as gnb

from IPython.display import display, Math, Latex
```

1 Exercice 5 : détection de fraude à la carte de crédit

from Roland Donat

Chaque année, les banques perdent d'importantes sommes d'argents suite aux pertes ou aux vols de cartes de crédit. Pour détecter l'occurrence de fraudes, l'industrie financière se tourne de plus en plus vers l'intelligence artificielle et l'analyse de données pour trouver des solutions à cette problématique. En effet, les propriétaires de carte de crédit ont tendance à faire leurs achats en suivant des schémas plus ou moins identifiables. Quand ce schéma n'est plus respecté, il y a de fortes chances qu'une fraude se soit produite. Les paragraphes suivants donnent des informations générales sur le comportement des propriétaires de carte :

- Quand un propriétaire de carte de crédit voyage à l'étranger, les transactions frauduleuses sont plus probables car les touristes sont des cibles privilégiées pour les voleurs. Plus précisément, on estime que 1% des transactions sont frauduleuses quand le propriétaire de la carte de crédit est en voyage, contre seulement 0.2% de transactions frauduleuses à domicile. En moyenne, 5% de toutes les transactions se produisent au cours de voyages à l'étranger.
- Si une transaction est frauduleuse, alors la probabilité qu'il s'agisse d'un achat à l'étranger augmente, sauf si le propriétaire de la carte est précisément en voyage à l'étranger. Statistiquement, quand le propriétaire de la carte n'est pas en voyage, 10% des transactions frauduleuse concernent un achat à l'étranger alors que seulement 1% des transactions normales portent sur un achat à l'étranger. En revanche, quand le propriétaire est en voyage à l'étranger, 90% des transactions (frauduleuses ou non) sont des achats à l'étranger.
- Les achats réalisés sur internet sont plus souvent frauduleux. Ceci est particulièrement vrai pour les propriétaires de carte n'ayant pas d'ordinateur. En effet :
  - Pour ceux qui ne possèdent pas d'ordinateur, seulement 0.1% de leurs transactions normales sont faites sur internet. Ce chiffre monte à 1.1% en cas de transactions frauduleuses.
  - Pour les possesseurs d'ordinateur, 1% des transactions normales a lieu sur internet. En revanche, ce pourcentage s'élève à 2% lors de transactions frauduleuses.
  - On estime aujourd'hui que 75% de la population possède un ordinateur
- Malheureusement, les banques ne savent pas si le propriétaire d'une carte possède un ordinateur. Toutefois, ces dernières peuvent vérifier l'historique des transactions afin de rechercher si

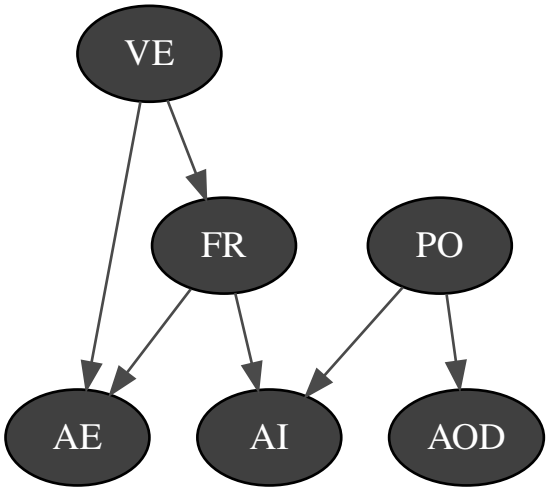
des achats liés à du matériel informatique ont été effectués dernièrement. En particulier, on estime que 10% des propriétaires d'ordinateur ont fait des achats en rapport avec l'informatique dernièrement contre 0.1% pour ceux ne possédant pas d'ordinateur.

Construire un BN (graphe et TPC) visant à détecter des transactions frauduleuses. Le BN sera composé des 6 variables booléennes suivantes :

```
FR : la transaction courante est frauduleuse.
VE : le propriétaire de la carte est actuellement en voyage à l'étranger.
AE : la transaction courante concerne un achat à l'étranger.
AI : la transaction courante concerne un achat sur internet.
PO : le propriétaire de la carte a un ordinateur.
AOD : un achat lié à l'informatique a été effectué dernièrement.
```

[2]:
```
bn=gum.BayesNet()

fr = bn.add(gum.LabelizedVariable("FR", "Transaction courante frauduleuse ?",␣
 ↪0).addLabel("non").addLabel("oui"))
ve = bn.add(gum.LabelizedVariable("VE", "Propriétaire CB à l'étranger ?", 0).
 ↪addLabel("non").addLabel("oui"))
ae = bn.add(gum.LabelizedVariable("AE", "Transaction courante concerne un achat␣
 ↪à l'étranger ?", 0).addLabel("non").addLabel("oui"))
ai = bn.add(gum.LabelizedVariable("AI", "Transaction courante concerne un achat␣
 ↪sur internet ?", 0).addLabel("non").addLabel("oui"))
po = bn.add(gum.LabelizedVariable("PO", "Propriétaire CB possède un PC ?", 0).
 ↪addLabel("non").addLabel("oui"))
aod = bn.add(gum.LabelizedVariable("AOD", "Achat informatique récent ?", 0).
 ↪addLabel("non").addLabel("oui"))

bn.addArc(ve, ae)
bn.addArc(ve, fr)
bn.addArc(fr, ae)
bn.addArc(fr, ai)
bn.addArc(po, ai)
bn.addArc(po, aod)

bn.cpt(ve)[:] = [0.95, 0.05]

bn.cpt(fr)[{"VE":"non"}] = [0.998, 0.002]
bn.cpt(fr)[{"VE":"oui"}] = [0.99, 0.01]

bn.cpt(ae)[{"VE":"non", "FR":"non"}] = [0.99, 0.01]
bn.cpt(ae)[{"VE":"non", "FR":"oui"}] = [0.9, 0.1]
bn.cpt(ae)[{"VE":"oui", "FR":"non"}] = [0.1, 0.9]
bn.cpt(ae)[{"VE":"oui", "FR":"oui"}] = [0.1, 0.9]

bn.cpt(ai)[{"FR":"non", "PO":"non"}] = [0.99, 0.01]
bn.cpt(ai)[{"FR":"non", "PO":"oui"}] = [0.9, 0.1]
```

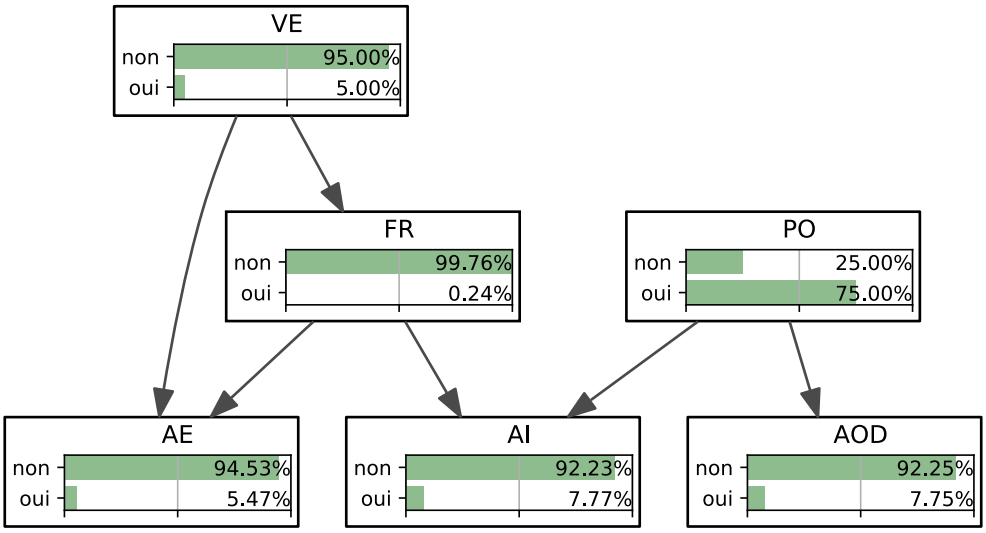
```
bn.cpt(ai)[{"FR":"oui", "PO":"non"}] = [0.89, 0.11]
bn.cpt(ai)[{"FR":"oui", "PO":"oui"}] = [0.8, 0.2]

bn.cpt(po)[:] = [0.25, 0.75]

bn.cpt(aod)[{"PO":"non"}] = [0.99, 0.01]
bn.cpt(aod)[{"PO":"oui"}] = [0.9, 0.1]

gnb.showBN(bn)
```



1.1 Inférence

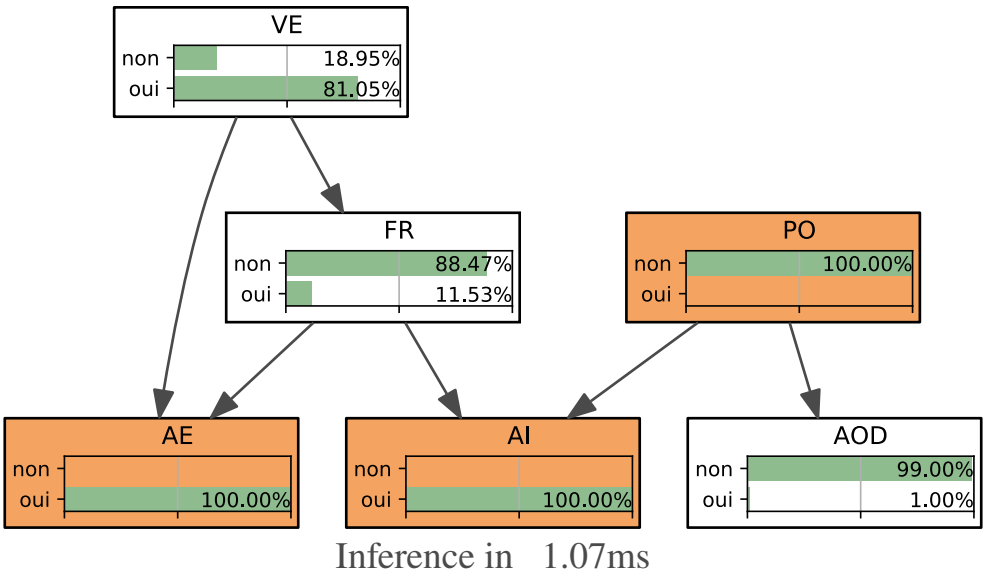
Quelle est la probabilité a priori qu'une transaction soit frauduleuse?

[3]:
```
gnb.showInference(bn)

p=gum.getPosterior(bn,target="FR",evs={})
print("La probabilité d'une transaction frauduleuse est ici {:4.2f}%".
 ↪format(100*p[{'FR':'oui'}]))
```



La probabilité d'une transaction frauduleuse est ici 0.24%

[4]:
```
gnb.showInference(bn,evs={"AE":"oui", "AI":"oui", "PO":"non"})

p=gum.getPosterior(bn,target="FR",evs={"AE":"oui", "AI":"oui", "PO":"non"})
print("La probabilité d'une transaction frauduleuse est ici {:4.2f}%".
 ↪format(100*p[{'FR':'oui'}]))
```



```
La probabilité d'une transaction frauduleuse est ici 11.53%
```

1.2 Première optimisation

Supposons à présent que vous ayez volé une carte de crédit - attention c'est mal ! -. Supposons de plus que vous connaissez les réseaux bayésiens et que vous savez que la banque du propriétaire de la carte utilise le système de détection de fraudes reposant sur le RB décrit précédemment. Malgré tout cela, vous souhaitez quand même faire un achat sur internet avec la carte volée, quelle(s) action(s) pouvez vous effectuer afin de réduire le risque que votre transaction soit rejetée? De combien pouvez vous faire baisser la probabilité que la transaction soit considérée comme frauduleuse ?

[5]:
```
p=gum.getPosterior(bn,target="FR",evs={"AI":"oui"})
print("La probabilité d'une transaction frauduleuse sur internet est {:4.2f}%".
 ↪format(100*p[{'FR':'oui'}]))

p=gum.getPosterior(bn,target="FR",evs={"AI":"oui","AE":"non"})
print("La probabilité d'une transaction frauduleuse sur internet en France est␣
 ↪{:4.2f}%".format(100*p[{'FR':'oui'}]))

p=gum.getPosterior(bn,target="FR",evs={"AI":"oui","AE":"oui"})
print("La probabilité d'une transaction frauduleuse sur internet à l'étranger␣
 ↪est {:4.2f}%".format(100*p[{'FR':'oui'}]))
```

```
La probabilité d'une transaction frauduleuse sur internet est 0.55%
La probabilité d'une transaction frauduleuse sur internet en France est 0.43%
La probabilité d'une transaction frauduleuse sur internet à l'étranger est 2.64%
```

[ ]:
```
```

06-ModelSelection

September 23, 2019

[1]:
```
import pyAgrum as gum
import pyAgrum.lib.notebook as gnb
from IPython.display import display, Math, Latex
```

1 Exercice 6

From Roland Donat

L'objectif de cet exercice est de comparer l'influence du choix de la structure d'un RB sur la représentativité de données observées.

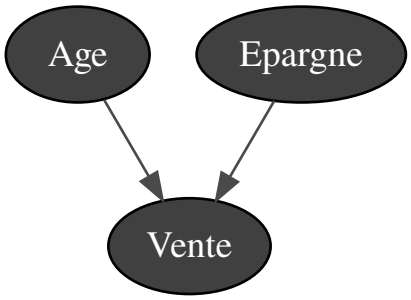
Pour ce faire, nous nous intéresserons à des données de retour marketing sur la vente de livret A. Dans ces données, les individus sont décrits par les variables suivantes :

```
(A)ge : classe d'âges de l'individu - valeurs possibles : [18,25], [26,59], 60+;
(E)pargne : l'individu a t-il de l'épargne - valeurs possibles : non, oui;
(V)ente_livret_A : vente d'un livret A à l'individu - valeurs possibles : échec, succès.
```

1.1 Modèle 1 : $A \rightarrow V \leftarrow E$

Construire un modèle pour le BN.

[2]:
```
bn1=gum.BayesNet("Vente livret A - modèle 1")

# on ne s'occupe pas des modalités
a1=bn1.add("Age",3)
v1=bn1.add("Vente",2)
e1=bn1.add("Epargne",2)

bn1.addArc(a1,v1)
bn1.addArc(e1,v1)

gnb.showBN(bn1)
```



Lancer l'apprentissage des paramètres sur le fichier "livretA_10000.csv". Indiquer les distributions a priori que propose cette apprentissage.

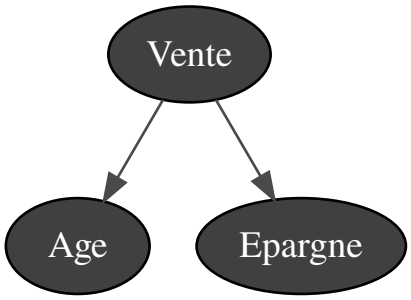
[14]:
```
learner=gum.BNLearner("livretA_10000.csv")
learn1=learner.learnParameters(bn1.dag())

gnb.sideBySide(*[learn1.cpt(i) for i in learn1.nodes()])
```

```
<IPython.core.display.HTML object>
```

1.2 Modèle 2 : $A \leftarrow V \rightarrow E$

Même question que précédemment

[9]:
```
bn2=gum.BayesNet("Vente livret A - modèle 2")

# on ne s'occupe pas des modalités
a2=bn2.add("Age",3)
v2=bn2.add("Vente",2)
e2=bn2.add("Epargne",2)

bn2.addArc(v2,a2)
bn2.addArc(v2,e2)

gnb.showBN(bn2)
```



[16]:
```
learner=gum.BNLearner("livretA_10000.csv")
learn2=learner.learnParameters(bn2.dag())

gnb.sideBySide(*[learn1.cpt(i) for i in learn1.nodes()])
```

```
<IPython.core.display.HTML object>
```

1.3 Comparaison des 2 modèles

[17]:
```
gnb.sideBySide(gnb.getInference(learn1),gnb.getInference(learn2),
               captions=['premier modèle','second modèle'])
```

```
<IPython.core.display.HTML object>
```

[18]:
```
gnb.sideBySide(gnb.getInference(learn1,evs={"Age":1}),gnb.
 ↪getInference(learn2,evs={"Age":1}),
               captions=['premier modèle','second modèle'])
```

```
<IPython.core.display.HTML object>
```

1.3.1 Quel est le "meilleur" modèle ?

Utiliser la log-vraisemblance pour comparer les 2 modèles

$$ll = \sum_{d \in Base} \log P(d) = \sum_{d \in Base} \sum_{v \in BN} \log P(v_d | parents_d(v))$$

[23]:
```
#En utilisant Pandas  (un peu bulldozer mais bon ... :))

from math import fsum, log2 # Quelques fonctions mathématiques
import pandas # Module pour la manipulation des jeux de données

dataframe = pandas.read_csv('livretA_10000.csv')

ll1 = fsum([log2(learn1.cpt(i)[d]) for i in range(learn1.size()) for d in␣
 ↪dataframe.to_dict("records")])
ll2 = fsum([log2(learn2.cpt(i)[d]) for i in range(learn2.size()) for d in␣
 ↪dataframe.to_dict("records")])

print("LL pour modèle 1 {} (en moyenne par données : {})".format(ll1,ll1/
 ↪len(dataframe)))
print("LL pour modèle 2 {} (en moyenne par données : {})".format(ll2,ll2/
 ↪len(dataframe)))
```

```
LL pour modèle 1 -32787.90425601868 (en moyenne par données :
-3.278790425601868)
LL pour modèle 2 -33389.12971011179 (en moyenne par données :
-3.3389129710111787)
```

[27]:
```
# en utilisant pyAgrum.lib (tellement plus fin :-) )
import pyAgrum.lib.bn2scores as bnsc
print("Scores for modèle 1")
print(bnsc.computeScores(learn1,"livretA_10000.csv"))
print("Scores for modèle 2")
print(bnsc.computeScores(learn2,"livretA_10000.csv"))
```

```
Scores for modèle 1
(100.0, {'likelihood': -32787.90425601824, 'aic': -32796.90425601824, 'aicc':
-65593.82652644801, 'bic': -32907.49366743419, 'mdl': -33102.47968077734})
Scores for modèle 2
(100.0, {'likelihood': -33389.12971011014, 'aic': -33396.12971011014, 'aicc':
-66792.27062694432, 'bic': -33482.14369676699, 'mdl': -33639.70513486923})
```

[ ]:
```

```

07-ClassificationSupervise

September 23, 2019

[1]:
```
import pyAgrum as gum
import pyAgrum.lib.notebook as gnb

from IPython.display import display, Math, Latex

import pandas
import numpy as np
```

1 Exercice 7

From Roland Donnat

L'objectif de cet exercice est d'utiliser un RB afin de prédire le résultat de matchs de football de Ligue 1 (division 1 de football en France) à partir de certaines statistiques de jeu.

Deux jeux de données vont être utilisés :

- Un jeu de données d'apprentissage permettant de calibrer le modèle prédictif contanant l'historique des matchs des saisons 2005-2006 à 2009-2010 - Fichier fra_l1_app.csv
- Un jeu de données de test permettant d'évaluer le modèle contanant les matchs des saisons 2010-2011 et 2011-201 - Fichier fra_l1_test.csv

1.0.1 Prétraitement des données

Pour un apprentissage "facile", nous allons : + transformer toutes les données en entier ou chaîne, + s'assurer que la base de tests ne possèdent pas de valeurs inconnues pour la base d'apprentissage + sélectionner les variables "HomeTeam","AwayTeam","HST","AST" qui nous serviront à tenter de prédire "FTR" + créer les fichiers "learn.csv" et "test.csv" épurés.

[2]:
```
# data file is in the ../resources folder

datas=pandas.read_csv("fra_l1_app.csv")
tests=pandas.read_csv("fra_l1_test.csv")
```

[3]:
```
datas.head()
```

[3]:

	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	HS	\
0	2005-07-29	Paris SG	Metz	4	1	H	2	0	H	16.0	
1	2005-07-30	Marseille	Bordeaux	0	2	A	0	1	A	7.0	

```
2  2005-07-30      Nancy    Monaco     0     1   A     0     0   D  10.0
3  2005-07-30     Nantes      Lens     2     0   H     2     0   H  10.0
4  2005-07-30       Nice    Troyes     1     1   D     1     0   H  13.0

     AS  HST  AST  HY  AY  HR  AR
0  11.0  9.0  7.0   2   3   0   0
1   9.0  3.0  4.0   2   4   1   0
2   5.0  6.0  3.0   2   2   2   0
3  10.0  3.0  3.0   1   3   0   0
4   5.0  6.0  5.0   2   2   0   1
```

[4]:
```
tests.head()
```

[4]:
```
         Date   HomeTeam      AwayTeam  FTHG  FTAG FTR  HTHG  HTAG HTR    HS  \
0  2010-08-07       Lyon        Monaco     0     0   D     0     0   D  16.0
1  2010-08-07  Marseille          Caen     1     2   A     0     0   D  16.0
2  2010-08-07       Nice  Valenciennes     0     0   D     0     0   D  10.0
3  2010-08-07   Paris SG    St Etienne     3     1   H     2     1   H  19.0
4  2010-08-07     Rennes         Lille     1     1   D     1     0   H  20.0

     AS  HST  AST  HY  AY  HR  AR
0   9.0  5.0  4.0   2   3   0   0
1  12.0  3.0  2.0   2   3   0   0
2  11.0  2.0  1.0   0   3   0   0
3  14.0  6.0  5.0   2   1   0   0
4  14.0  5.0  5.0   3   3   0   0
```

1.0.2 Prétraitement

L'apprentissage avec pyAgrum fonctionne sur des csv totalement prétraités (des entiers). Il s'agit donc de préparer la base en ce sens.

[5]:
```
datas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1899 entries, 0 to 1898
Data columns (total 17 columns):
Date        1899 non-null object
HomeTeam    1899 non-null object
AwayTeam    1899 non-null object
FTHG        1899 non-null int64
FTAG        1899 non-null int64
FTR         1899 non-null object
HTHG        1899 non-null int64
HTAG        1899 non-null int64
HTR         1899 non-null object
HS          1899 non-null float64
AS          1899 non-null float64
```

```
HST          1899 non-null float64
AST          1899 non-null float64
HY           1899 non-null int64
AY           1899 non-null int64
HR           1899 non-null int64
AR           1899 non-null int64
dtypes: float64(4), int64(8), object(5)
memory usage: 252.3+ KB
```

[6]:
```
#pretraitement

# float en int
datas["HS"] = datas["HS"].astype(int)
datas["AS"] = datas["AS"].astype(int)
tests["HS"] = tests["HS"].astype(int)
tests["AS"] = tests["AS"].astype(int)
datas["HST"] = datas["HST"].astype(int)
datas["AST"] = datas["AST"].astype(int)
tests["HST"] = tests["HST"].astype(int)
tests["AST"] = tests["AST"].astype(int)

# s'assurer que les valeurs de tests sont incluses dans les valeurs␣
 ↪d'apprentissage
HST_app_max = np.max(datas["HST"])
AST_app_max = np.max(datas["AST"])
tests.loc[tests["HST"] > HST_app_max, "HST"] = HST_app_max
tests.loc[tests["AST"] > AST_app_max, "AST"] = AST_app_max

# ville -> entiers
towns = datas['HomeTeam'].unique()
towns2nums = dict(zip(towns,range(len(towns))))
nums2towns = dict(zip(range(len(towns)),towns))
datas['HomeTeamNum']=datas['HomeTeam'].replace(towns2nums)
datas['AwayTeamNum']=datas['AwayTeam'].replace(towns2nums)
tests['HomeTeamNum']=tests['HomeTeam'].replace(towns2nums)
tests['AwayTeamNum']=tests['AwayTeam'].replace(towns2nums)

# classe -> entiers
classes=datas['FTR'].unique()
classes2nums=dict(zip(classes,range(len(classes))))
datas['FTRNum']=datas['FTR'].replace(classes2nums)
tests['FTRNum']=tests['FTR'].replace(classes2nums)

datas.head()
```

[6]:

	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	HS	AS	\
0	2005-07-29	Paris SG	Metz	4	1	H	2	0	H	16	11	

```
1  2005-07-30  Marseille  Bordeaux     0     2   A     0     1   A   7   9
2  2005-07-30      Nancy    Monaco     0     1   A     0     0   D  10   5
3  2005-07-30     Nantes      Lens     2     0   H     2     0   H  10  10
4  2005-07-30       Nice    Troyes     1     1   D     1     0   H  13   5

   HST  AST  HY  AY  HR  AR  HomeTeamNum  AwayTeamNum  FTRNum
0    9    7   2   3   0   0            0           16       0
1    3    4   2   4   1   0            1           14       1
2    6    3   2   2   2   0            2           11       1
3    3    3   1   3   0   0            3           13       0
4    6    5   2   2   0   1            4           12       2
```

[7]:
```
tests.head()
```

[7]:
```
         Date   HomeTeam      AwayTeam  FTHG  FTAG FTR  HTHG  HTAG HTR  HS  \
0  2010-08-07       Lyon        Monaco     0     0   D     0     0   D  16
1  2010-08-07  Marseille          Caen     1     2   A     0     0   D  16
2  2010-08-07       Nice  Valenciennes     0     0   D     0     0   D  10
3  2010-08-07   Paris SG    St Etienne     3     1   H     2     1   H  19
4  2010-08-07     Rennes          Lille     1     1   D     1     0   H  20

   AS  HST  AST  HY  AY  HR  AR HomeTeamNum AwayTeamNum  FTRNum
0   9    5    4   2   3   0   0          17          11       2
1  12    3    2   2   3   0   0           1          23       1
2  11    2    1   0   3   0   0           4          21       2
3  14    6    5   2   1   0   0           0           6       0
4  14    5    5   3   3   0   0          18           8       2
```

Dans un second temps, nous décidons de nous limiter aux variables sivantes : + features : “HomeTeamNum”,“AwayTeamNum”,“HST”,“AST” + classe : “FTRNum”

[8]:
```
#variables selection
vars=["HomeTeamNum","AwayTeamNum","HST","AST","FTRNum"]

#sauvegarde des bases opérationnelles
datas[vars].to_csv("learn.csv",index=False)
tests[vars].to_csv("test.csv",index=False)
```

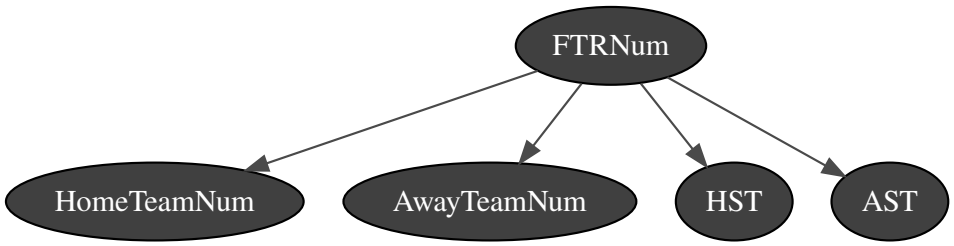
1.1 Modèle “Naive Bayes”

Nous essayons tout d’abord un modèle “Naïve Bayes” qui est connu pour ne pas être correct du tout théoriquement mais, finalement, pour assez bien fonctionner. Le principe du modèle est de supposer une structure du modèle où la variable à expliquer (la classe) est l’unique parents des variables explicantes (les features).

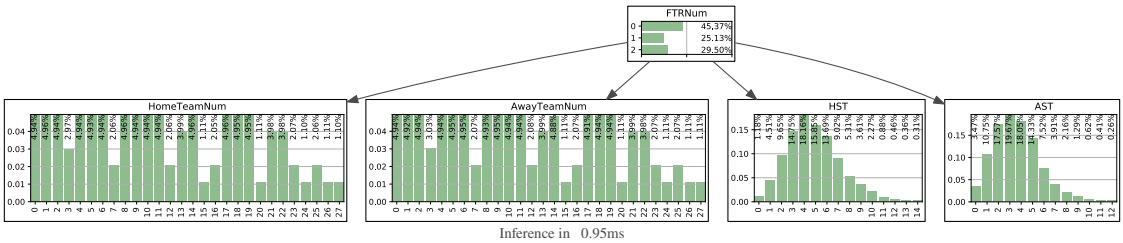
[9]:
```
naive=gum.BayesNet("Ligue 1")
```

```
def series_to_lv(series):
    # Crée une LabelizedVariable pour une colonne de dataframe
    labels = series.unique()
    labels.sort()
    lv = gum.LabelizedVariable(series.name, "", len(labels))
    if series.dtype == "O":
        for i in range(len(labels)):
            v.changeLabel(i,labels[i])
    print(lv)
    return lv

for va_name in vars:
  va = series_to_lv(datas[va_name])
  naive.add(va)

for va_name in vars[:-1]:
    naive.addArc(naive.idFromName(vars[-1]),naive.idFromName(va_name))

gnb.showBN(naive)
```

```
HomeTeamNum<0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27>
AwayTeamNum<0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27>
HST<0,1,2,3,4,5,6,7,8,9,10,11,12,13,14>
AST<0,1,2,3,4,5,6,7,8,9,10,11,12>
FTRNum<0,1,2>
```



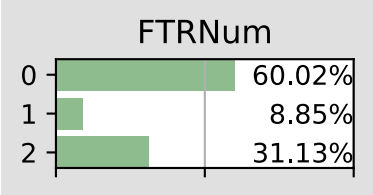
[11]:
```
learner=gum.BNLearner("learn.csv")
learner.useScoreBIC()
learner.useAprioriSmoothing(1)
naive=learner.learnParameters(naive.dag())

gnb.showInference(naive)
```



[12]:
```
print(nums2towns)
```

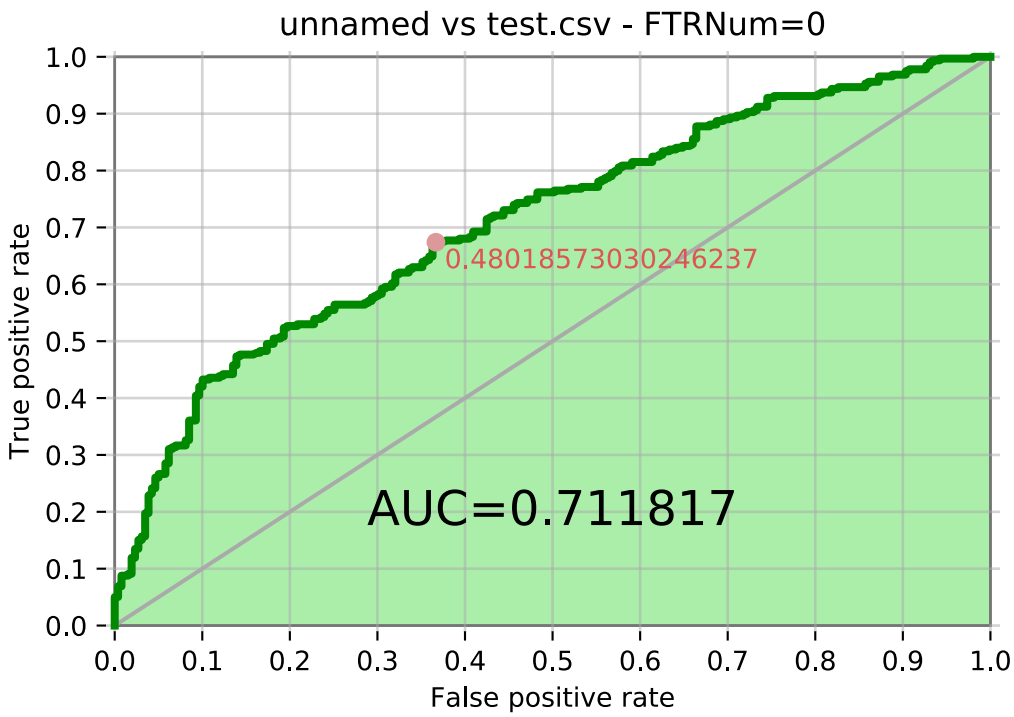
```
{0: 'Paris SG', 1: 'Marseille', 2: 'Nancy', 3: 'Nantes', 4: 'Nice', 5:
'Sochaux', 6: 'St Etienne', 7: 'Strasbourg', 8: 'Lille', 9: 'Le Mans', 10:
'Toulouse', 11: 'Monaco', 12: 'Troyes', 13: 'Lens', 14: 'Bordeaux', 15:
'Ajaccio', 16: 'Metz', 17: 'Lyon', 18: 'Rennes', 19: 'Auxerre', 20: 'Sedan', 21:
'Valenciennes', 22: 'Lorient', 23: 'Caen', 24: 'Le Havre', 25: 'Grenoble', 26:
'Montpellier', 27: 'Boulogne'}
```

[13]:
```
import matplotlib
%matplotlib inline
gnb.showPosterior(naive,target='FTRNum',evs={'HomeTeamNum':8,'AwayTeamNum':23})
```



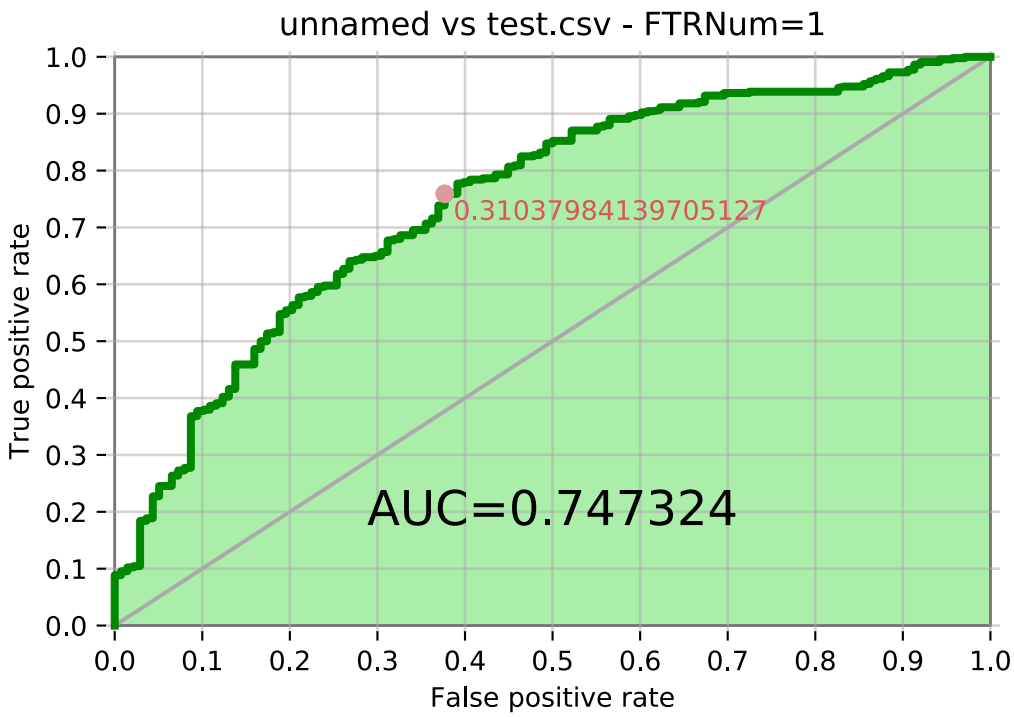
[14]:
```
import pyAgrum.lib.bn2roc as bn2roc

bn2roc.showROC(naive,"test.csv",variable="FTRNum",label="0")
```

```
 test.csv : [ ############################################################## ]
100%
 result in test.csv-ROC_unnamed-FTRNum-0.png
```



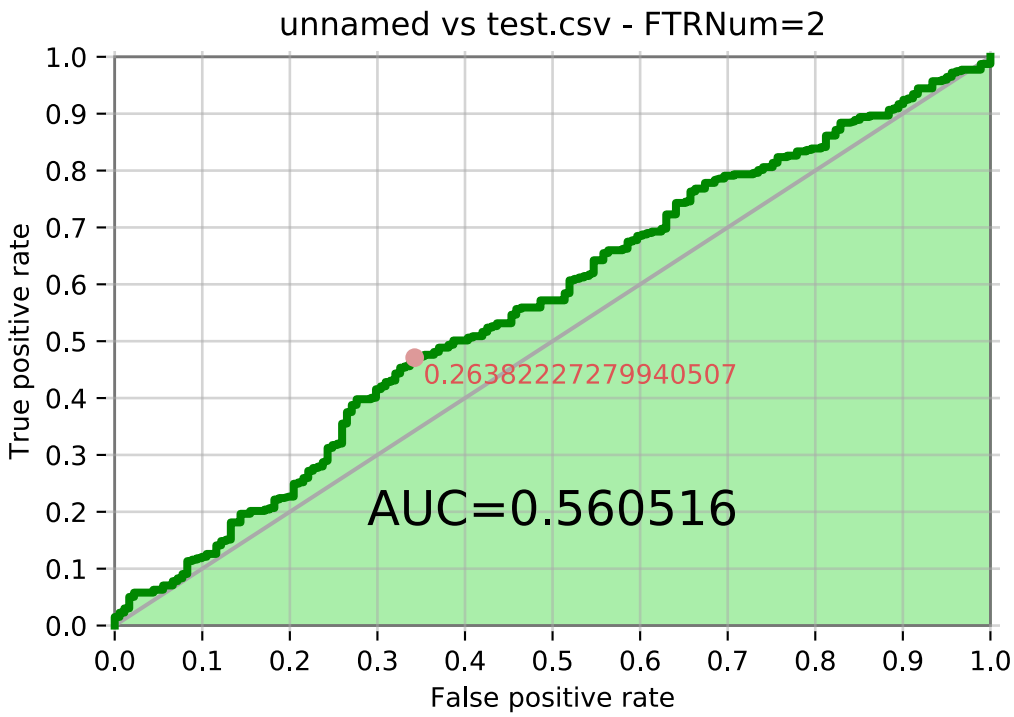
[15]:
```
bn2roc.showROC(naive,"test.csv",variable="FTRNum",label="1")
```

```
 test.csv : [ ############################################################## ]
100%
 result in test.csv-ROC_unnamed-FTRNum-1.png
```



[16]:
```
bn2roc.showROC(naive,"test.csv",variable="FTRNum",label="2")
```

```
 test.csv : [ ############################################################## ]
100%
 result in test.csv-ROC_unnamed-FTRNum-2.png
```

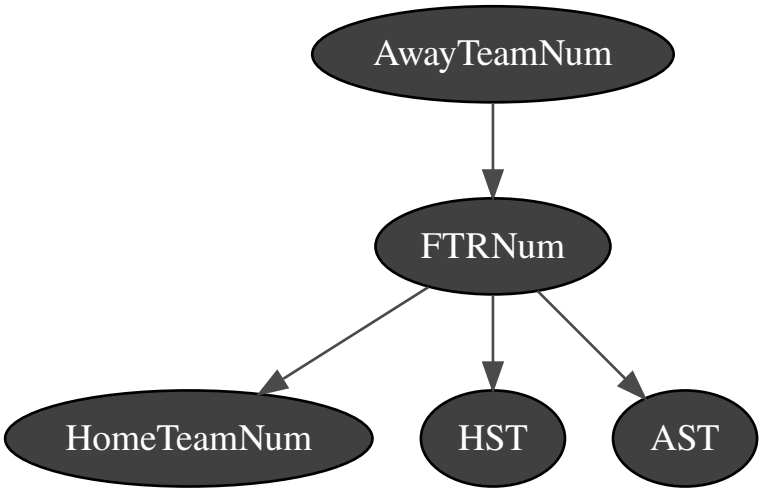


1.2 Apprentissage

Un apprentissage de structure peut-il donner mieux pour notre problème ?

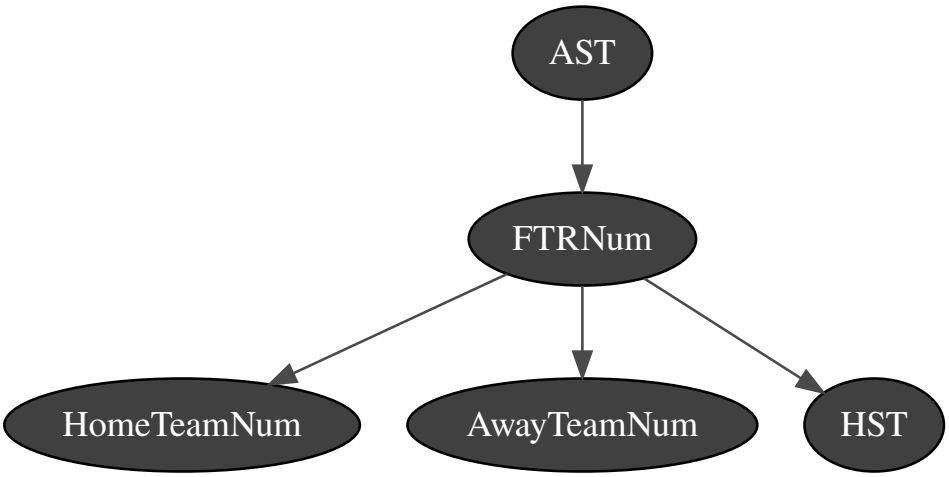
[17]:
```
learner=gum.BNLearner("learn.csv")
learner.useLocalSearchWithTabuList()
learner.useScoreAIC()
learner.useNoApriori()
model=learner.learnBN()
print("Learned in {0}s".format(learner.currentTime()))
gnb.showBN(model)
```

```
Learned in 0.005464s
```



[18]:
```
learner=gum.BNLearner("learn.csv")
learner.useGreedyHillClimbing()
learner.useScoreAIC()
learner.useNoApriori()
model=learner.learnBN()
print("Learned in {0}s".format(learner.currentTime()))
gnb.showBN(model)
```

```
Learned in 0.0016519999999999998s
```



08-dynamicBN

September 23, 2019

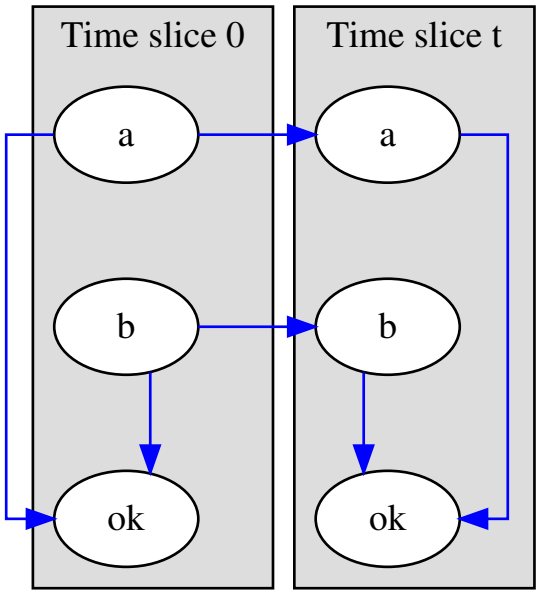
[1]:
```
import pyAgrum as gum
import pyAgrum.lib.notebook as gnb
import pyAgrum.lib.dynamicBN as gdyn

from math import exp
```

1 Exercice 8

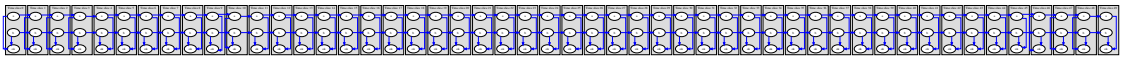
On se propose d'étudier la fiabilité et la maintenance d'un système décrit ainsi

[2]:
```
def fiabVar(s):
    return gum.LabelizedVariable(s,s,0).addLabel("NOK").addLabel("OK")

dbn=gum.BayesNet()
a0,b0,at,bt=[dbn.add(fiabVar(s)) for s in ["a0","b0","at","bt"]]

dbn.addArc(a0,at)
dbn.addArc(b0,bt)

ok0=dbn.addAND(fiabVar("ok0"))
dbn.addArc(a0,ok0)
dbn.addArc(b0,ok0)

okt=dbn.addAND(fiabVar("okt"))
dbn.addArc(at,okt)
dbn.addArc(bt,okt)
```

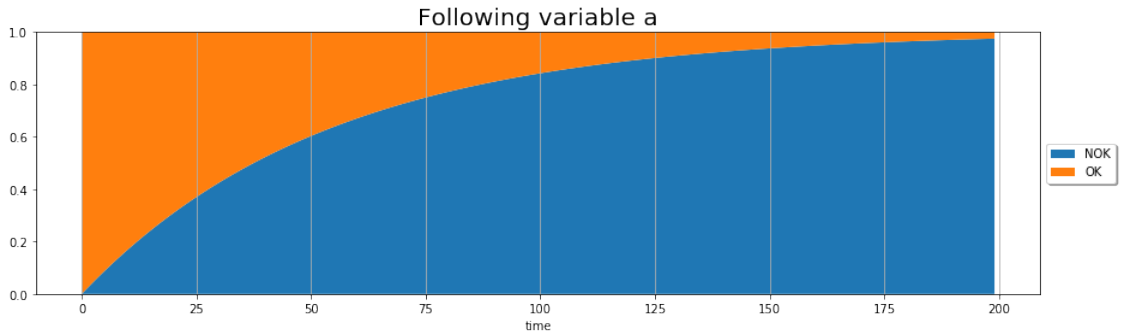
[4]:
```
mat = lambda x : [1,0,exp(-x),1-exp(-x)]

dbn.cpt(a0).fillWith([0,1])
dbn.cpt(b0).fillWith([0,1])

dbn.cpt(at).fillWith(mat(4))
dbn.cpt(bt).fillWith(mat(2))

gdyn.showTimeSlices(dbn)
```

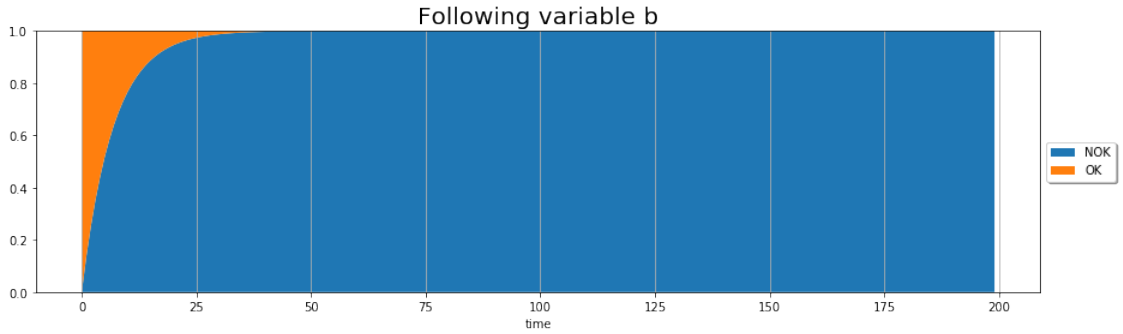


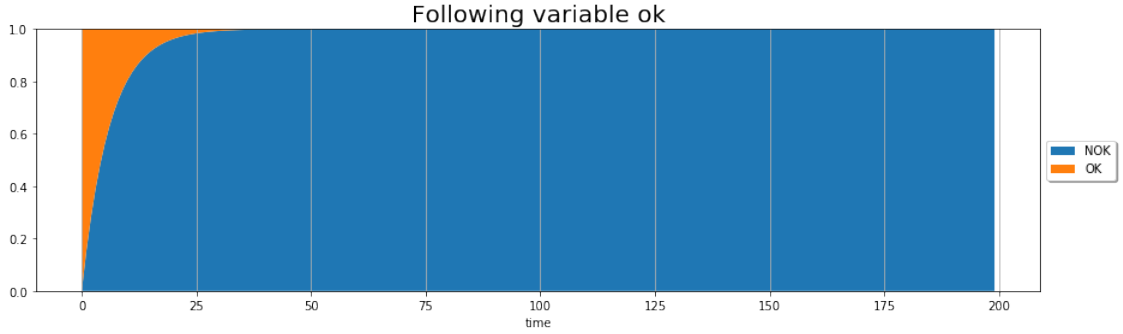
[5]:
```
Taille=50

bn=gdyn.unroll2TBN(dbn,Taille)
gdyn.showTimeSlices(bn,size="10")
```



[6]:
```
Taille=200

import matplotlib.pyplot as plt
%matplotlib inline

plt.rcParams['figure.figsize'] = (15, 4)
gdyn.plotFollow(["a","b","ok"],dbn,T=Taille,evs={})
```

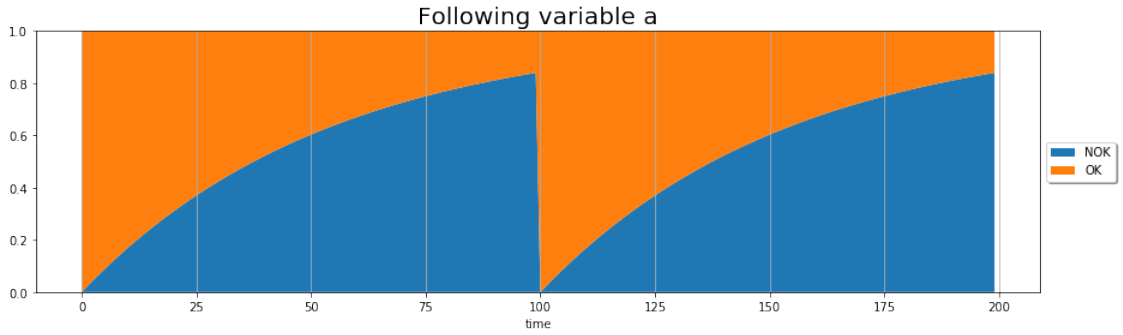


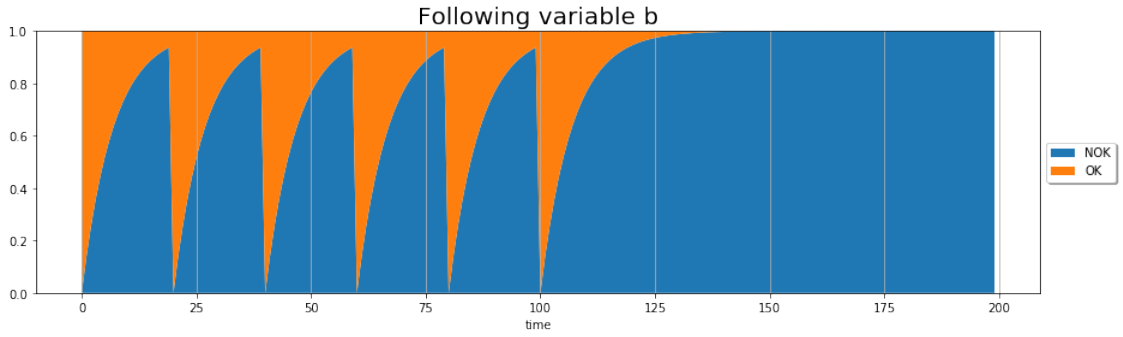


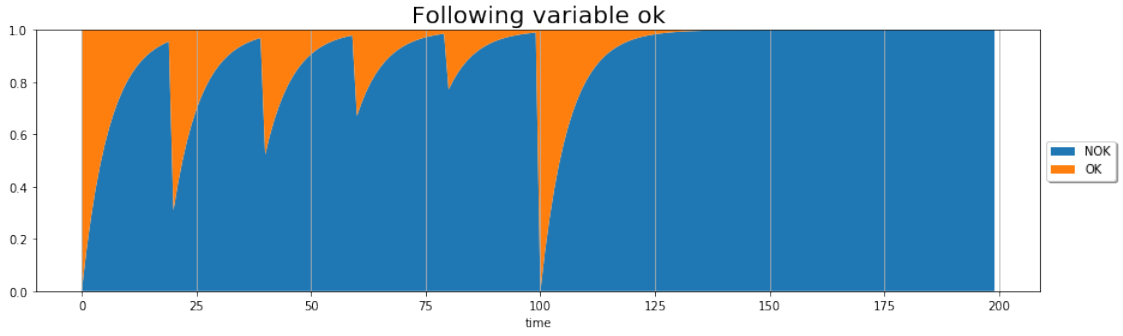


[10]:
```
bn=gdyn.unroll2TBN(dbn,Taille)

def repare(bn,var):
    idvar=bn.idFromName(var)
    for p in bn.parents(idvar):
        bn.eraseArc(p,idvar)
    bn.cpt(idvar).fillWith([0,1])

repare(bn,"b20")
repare(bn,"b40")
```

```
repare(bn,"b60")
repare(bn,"b80")
repare(bn,"b100")
repare(bn,"a100")

gdyn.plotFollowUnrolled(["a","b","ok"],bn,T=Taille,evs={})
```







[ ]: