Introduction, introspection, illustration

Pierre-Henri WUILLEMIN

LIP6
pierre-henri.wuillemin@lip6.fr

1. introduction
   - short history
   - components
   - opensource project
   - next

2. introspection : focus on 3 elementary Components

3. illustration
   - The model (Liessman Eric Sturlaugson, Montana, 2014)
     - dynamic Bayesian Network
     - Chaîne de Markov à temps continu
     - CTBN
   - Quick implementation of CTBNs using pyAgrum

aGrUM/pyAgrum

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).
4. Usable and improvable by others than us

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).
4. Usable and improvable by others than us (i.e. students).

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).
4. Usable and improvable by others than us (i.e. students).

- Goals 1 to 3 rather well achieved

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).
4. Usable and improvable by others than us (i.e. students).

- Goals 1 to 3 rather well achieved but 4 was (at least) questionnable.

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).
4. Usable and improvable by others than us (i.e. students).

- Goals 1 to 3 rather well achieved but 4 was (at least) questionnable.
- Goals 1 to 3 make aGrUM interesting enough for outside the laboratory.

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).
4. Usable and improvable by others than us (i.e. students).

- Goals 1 to 3 rather well achieved but 4 was (at least) questionnable.
- Goals 1 to 3 make aGrUM interesting enough for outside the laboratory.
- pyAgrum as a solution.

aGrUM/pyAgrum : a (very short) history

(> 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).
4. Usable and improvable by others than us (i.e. students).

- Goals 1 to 3 rather well achieved but 4 was (at least) questionnable.
- Goals 1 to 3 make aGrUM interesting enough for outside the laboratory.
- pyAgrum as a solution.

(< 6 year) pyAgrum's goals :

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).
4. Usable and improvable by others than us (i.e. students).

- Goals 1 to 3 rather well achieved but 4 was (at least) questionnable.
- Goals 1 to 3 make aGrUM interesting enough for outside the laboratory.
- pyAgrum as a solution.

($<$ 6 year) pyAgrum's goals :

1. Wrapper of aGrUM (package, optimized, etc.).

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).
4. Usable and improvable by others than us (i.e. students).

- Goals 1 to 3 rather well achieved but 4 was (at least) questionnable.
- Goals 1 to 3 make aGrUM interesting enough for outside the laboratory.
- pyAgrum as a solution.

($<$ 6 year) pyAgrum's goals :

1. Wrapper of aGrUM (package, optimized, etc.).
2. Useful, accessible and improvable for as many people as possible.

aGrUM/pyAgrum : a (very short) history

(> 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).
4. Usable and improvable by others than us (i.e. students).

- Goals 1 to 3 rather well achieved but 4 was (at least) questionnable.
- Goals 1 to 3 make aGrUM interesting enough for outside the laboratory.
- pyAgrum as a solution.

(< 6 year) pyAgrum's goals :

1. Wrapper of aGrUM (package, optimized, etc.).
2. Useful, accessible and improvable for as many people as possible.
3. Public, documented and deployed as widely as possible.

aGrUM/pyAgrum : a (very short) history

($>$ 10 years) aGrUM's goals (as a tool for laboratory)

1. PGM as a library(and not as a software $\Rightarrow$ No IDE).
2. C++ ! ! !
3. Optimized (as much as possible).
4. Usable and improvable by others than us (i.e. students).

- Goals 1 to 3 rather well achieved but 4 was (at least) questionnable.
- Goals 1 to 3 make aGrUM interesting enough for outside the laboratory.
- pyAgrum as a solution.

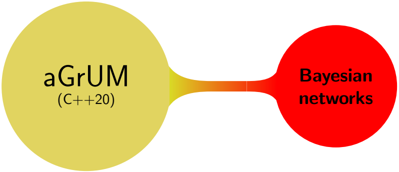
($<$ 6 year) pyAgrum's goals :

1. Wrapper of aGrUM (package, optimized, etc.).
2. Useful, accessible and improvable for as many people as possible.
3. Public, documented and deployed as widely as possible.

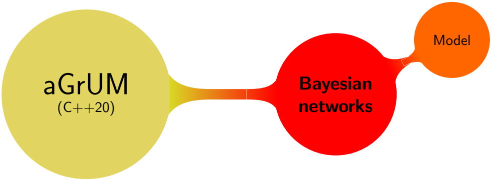
- repository : svn$\rightarrow$local git$\rightarrow$local gitlab$\rightarrow$gitlab
- Open Source : GPL then LGPL.

aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models

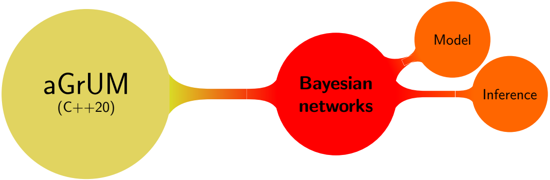
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



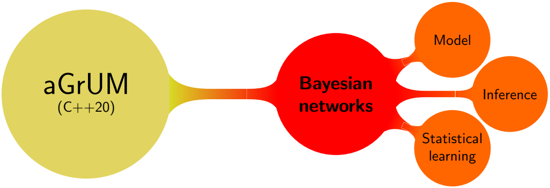
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



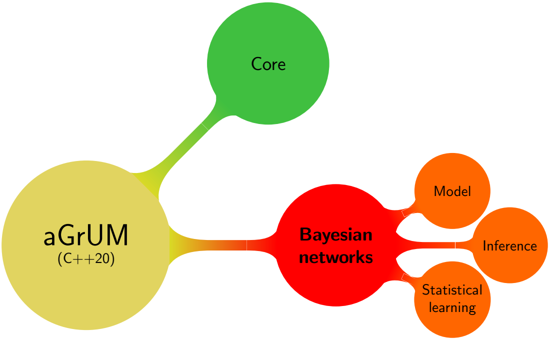
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



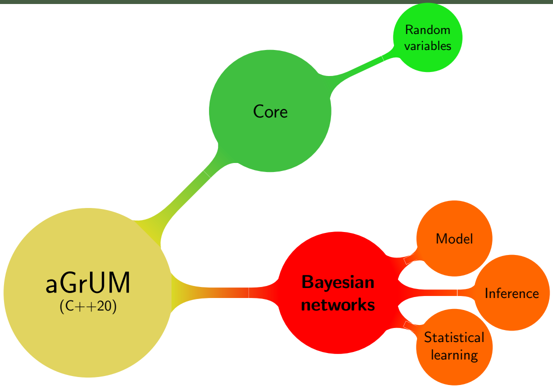
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



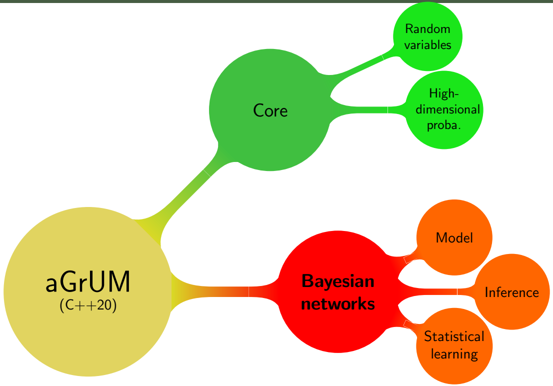
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



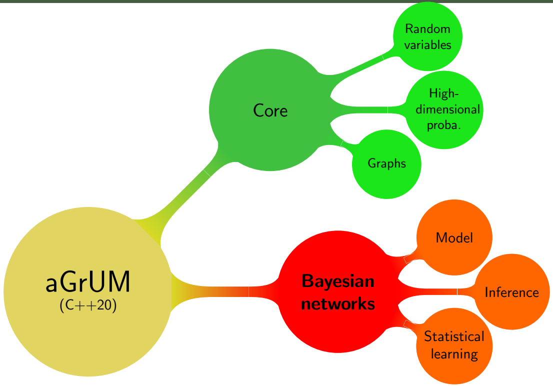
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



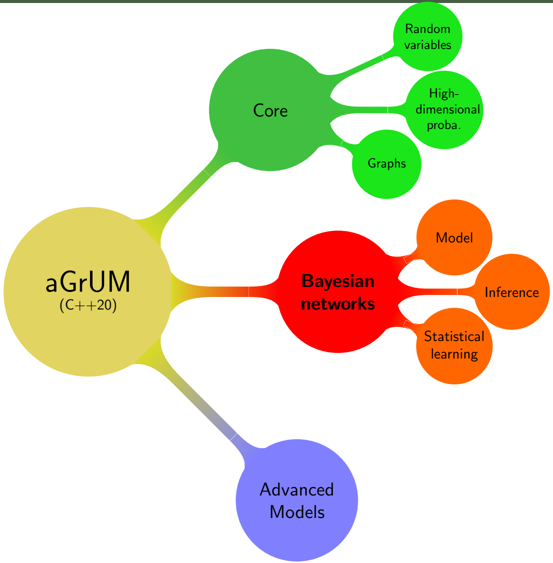
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



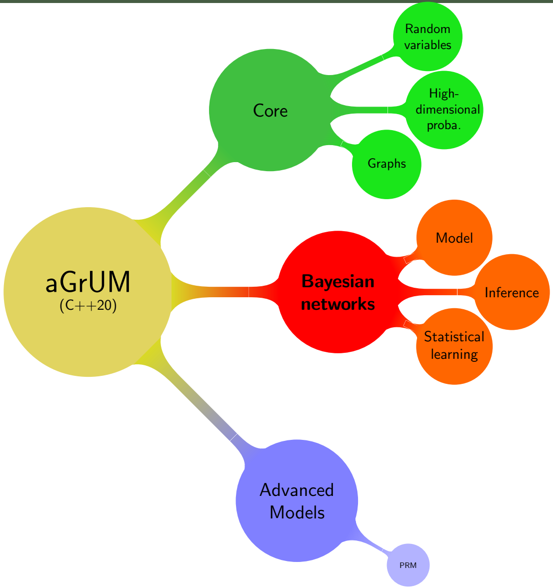
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



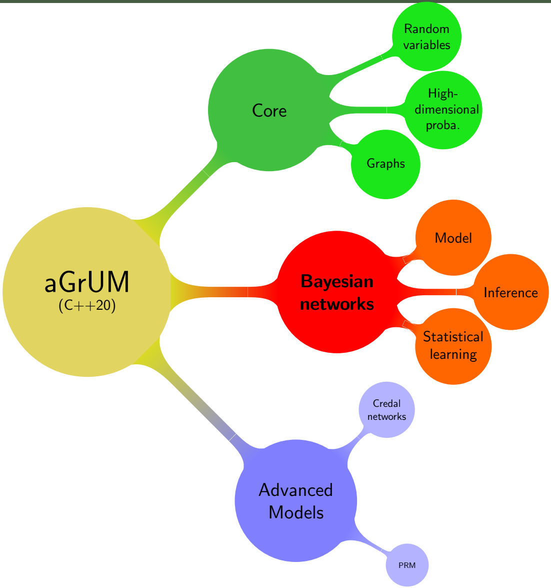
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



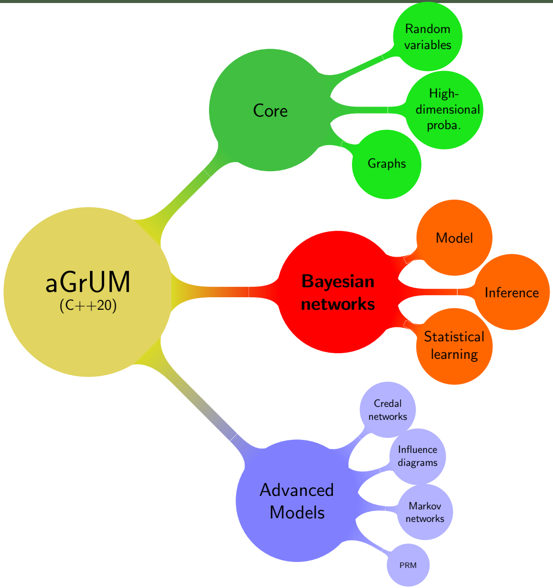
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



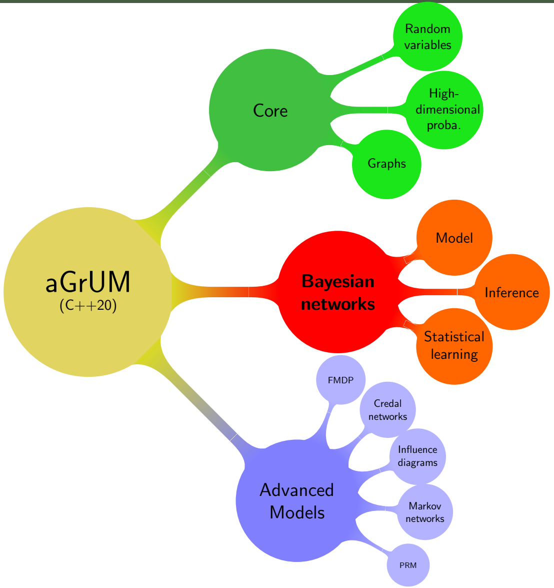
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



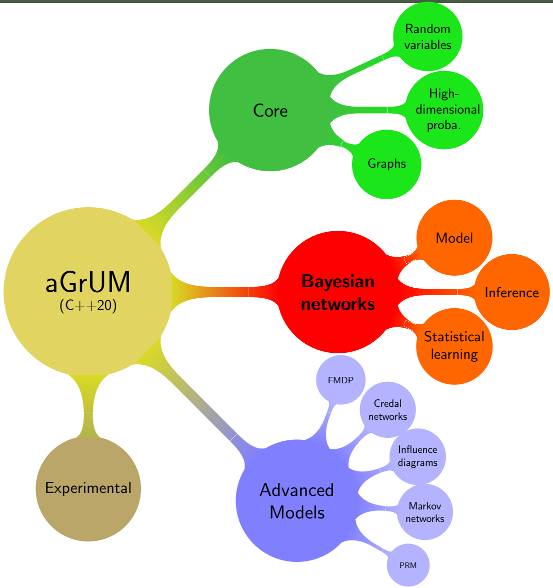
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



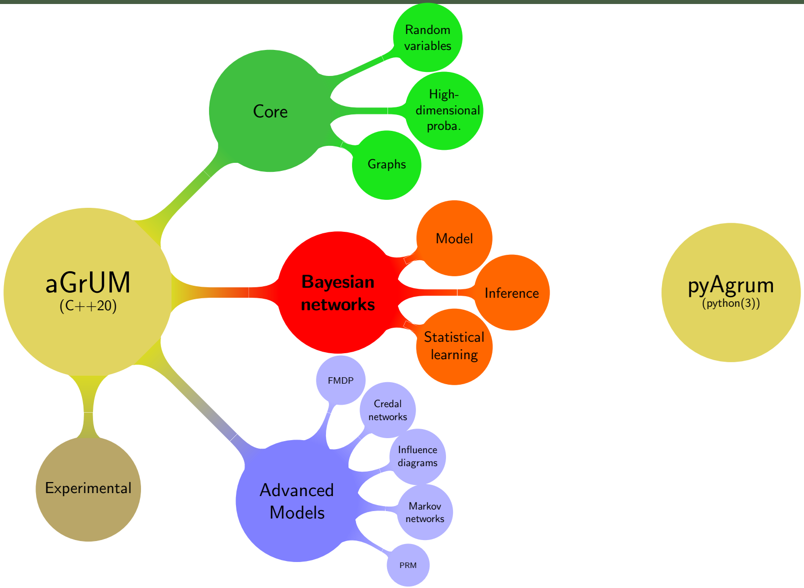
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



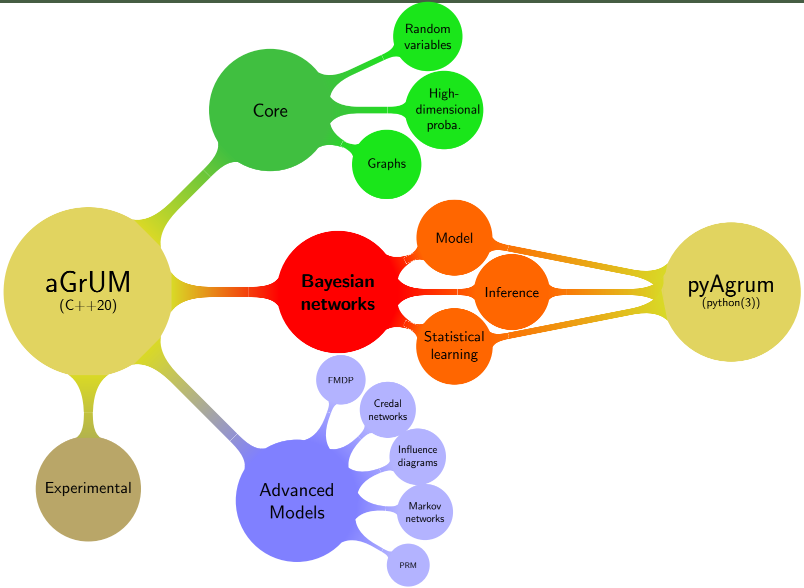
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



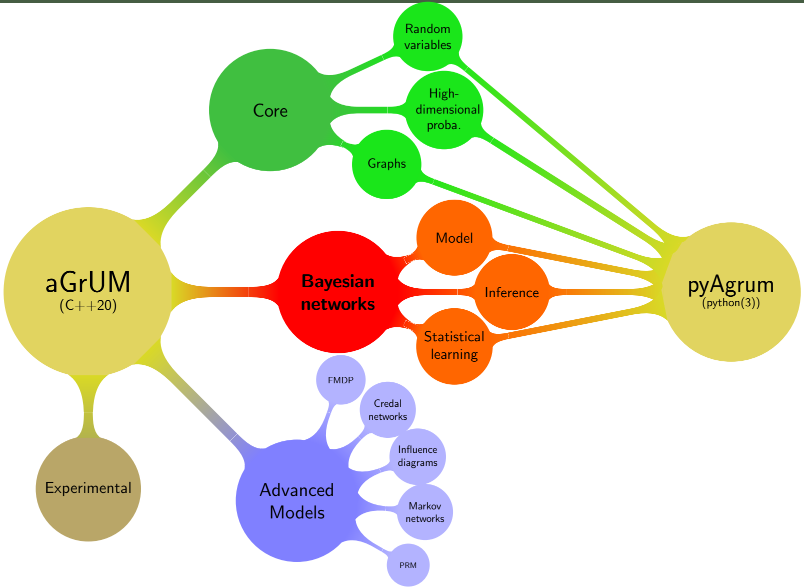
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



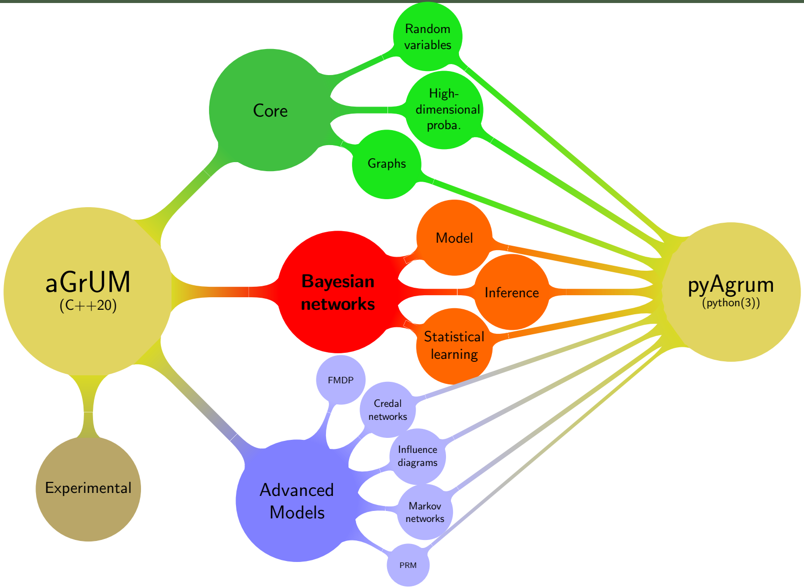
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



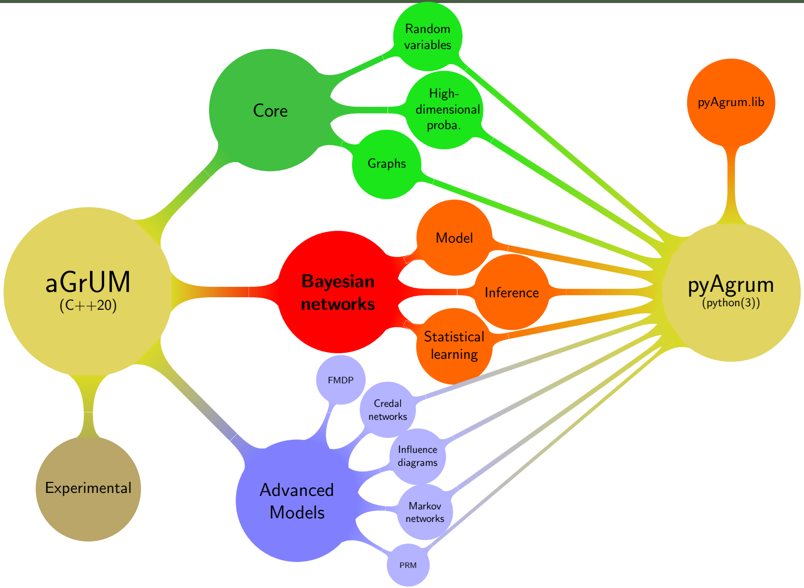
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



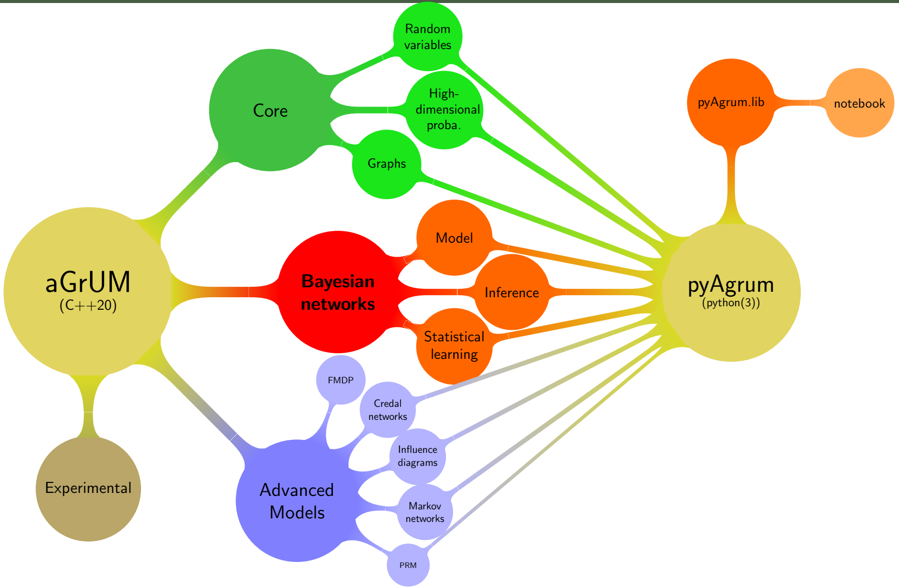
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



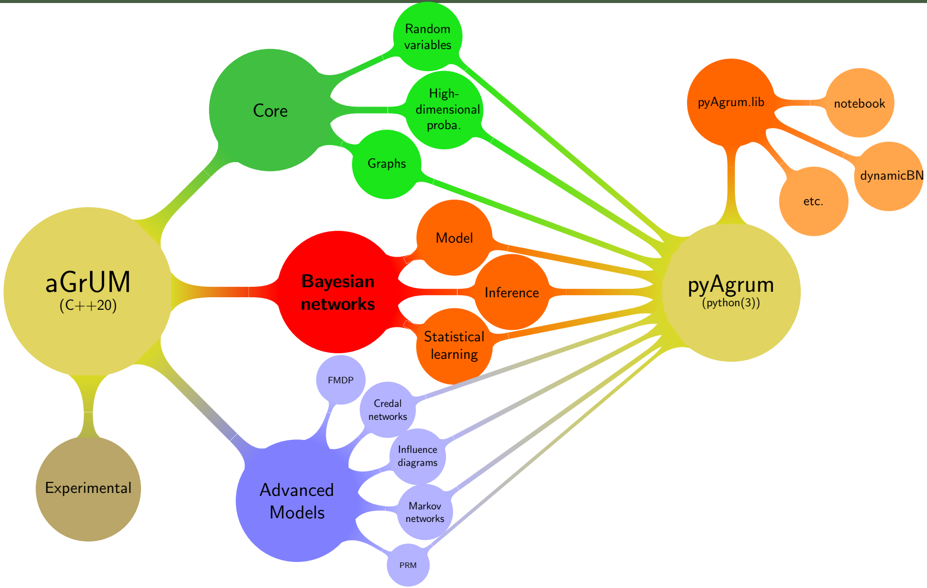
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



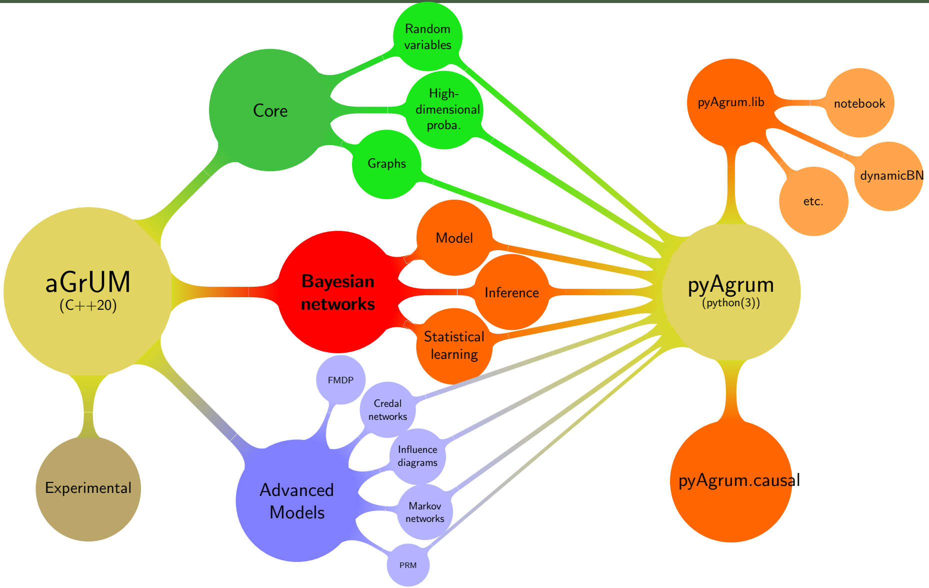
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models



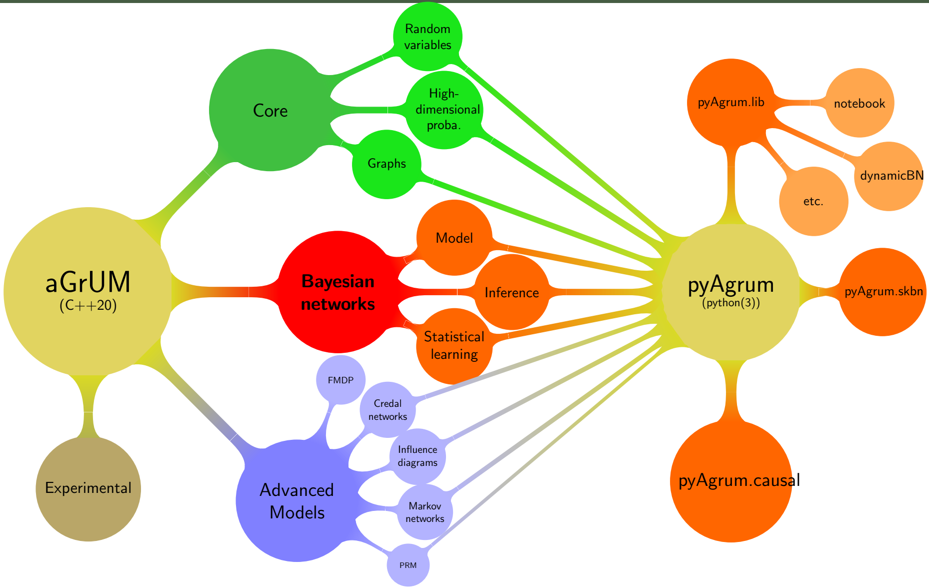
aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models

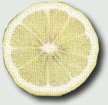


aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models

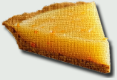


aGrUM/pyAgrum : a framework for modeling, learning and using (probabilistic) graphical models





aGrUM/pyAgrum as OpenSource project



Contents
- aGrUM
- pyAgrum

Documentations
- aGrUM's doc
- pyAgrum's ReadTheDocs

Notebooks
- pyAgrum's Tutorials
- From The Book Of Why

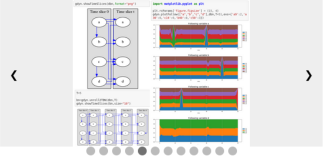
RECENT POSTS
- aGrUM/pyAgrum 0.22.8 released
- aGrUM/pyAgrum 0.22.7 released
- aGrUM/pyAgrum 0.22.6 released
- aGrum/pyAgrum in 2021

Stats on downloads

pypi package 0.22.8 | dwnlds 20.6k/month
Anaconda.org 0.22.8 | dwnlds 5.2k/month
aGrUM aGrUM aGrUM

aGrUM/pyAgrum

A GRaphical Universal Modeler (https://gitlab.com/agrumery/aGrUM)



aGrUM

aGrUM is a C++ library for graphical models. It is designed for easily building applications using graphical models such as Bayesian networks, influence diagrams, decision trees, GAI networks or Markov decision processes.

aGrUM is written to provide the basic building blocks to perform the following tasks :

- designing graphical models,
- learning graphical models,
- elicitation of graphical models,
- inference within graphical models,
- planification.

The probabilistic graphical models currently present in the library are the following:

- Bayesian networks (first and main target),
- Influence Diagrams,
- Markov networks,
- Credal networks,
- O3PRM (Probabilistic Relational Models).

pyAgrum

pyAgrum is a Python wrapper for the C++ aGrUM library (using SWIG interface generator). It provides a high-level interface to the part of aGrUM allowing to create, model, learn, use, calculate with and embed Bayesian Networks and other graphical models. Some specific (python and C++) codes are added in order to simplify and extend the aGrUM API.

Several topics have been added to pyAgrum (as pure python modules using pyAgrum) :

- Scikit-learn-compliant probabilistic classifiers based on Bayesian networks,
- Probabilistic causality (causal networks, do-calculus),
- dynamic Bayesian network,
- tools for explainability in Bayesian networks.

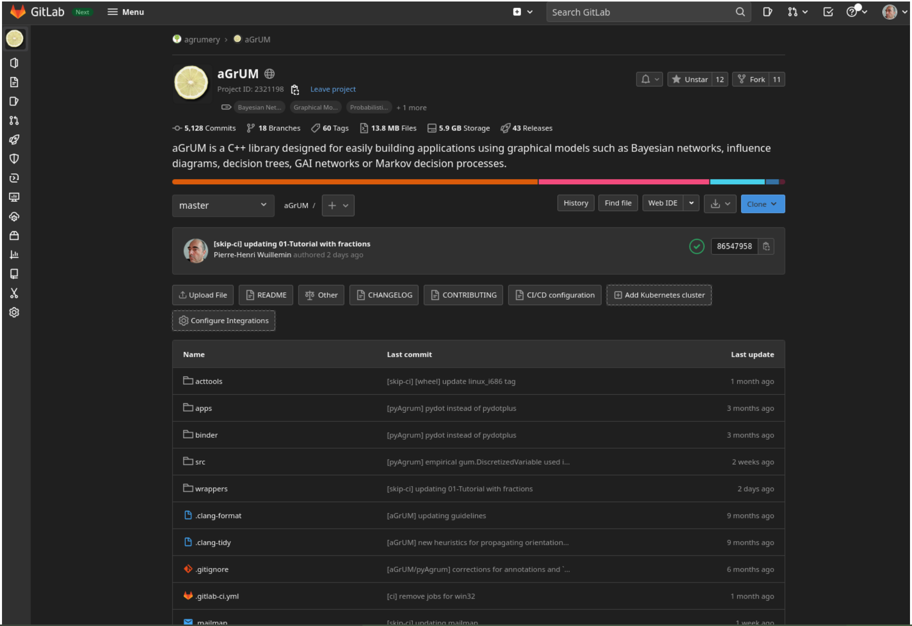
See the tutorials as jupyter notebooks for more details.

Installation : here

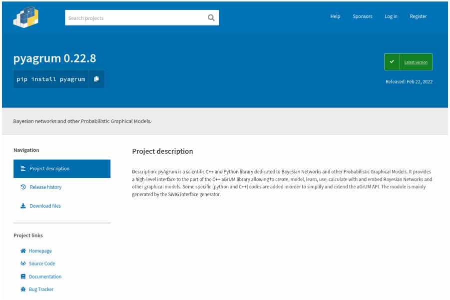
Licence

aGrUM/pyAgrum is released under the Gnu Lesser General Public License (LGPL v3.0), which means it can be freely copied and distributed, and costs nothing. Especially, aGrUM can be used and linked into both free software and proprietary software, provided that the code used under the LGPL is re-licensed under the LGPL (the other parts of the software are permitted have other licenses.). If you wish to integrate the aGrUM library into your product without being affected by the LGPL v.3, please contact us.

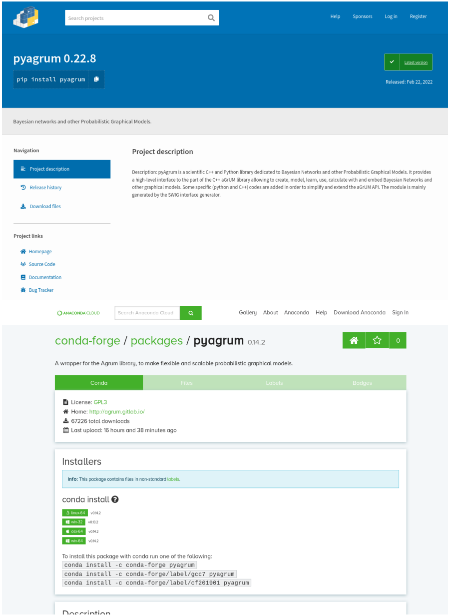
aGrUM/pyAgrum on gitlab.com



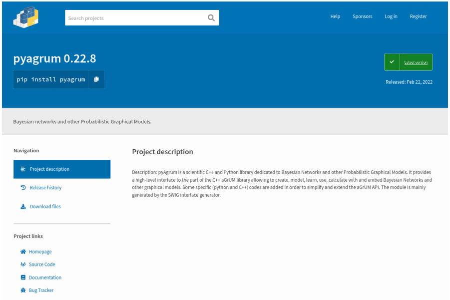
pyAgrum on pipy/anaconda/binder/readthedocs

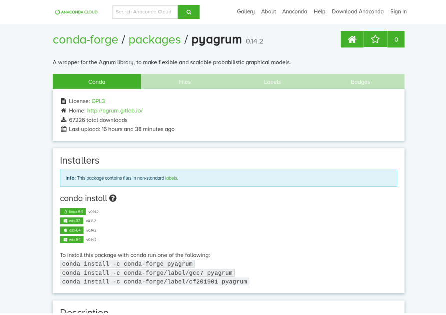


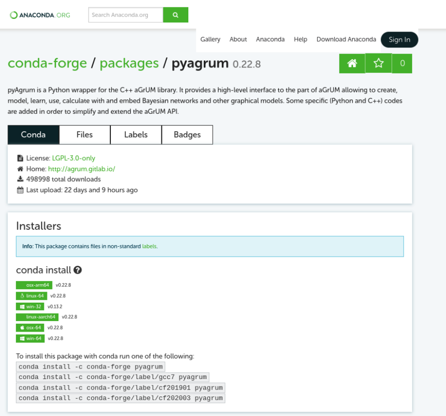
pyAgrum on pipy/anaconda/binder/readthedocs



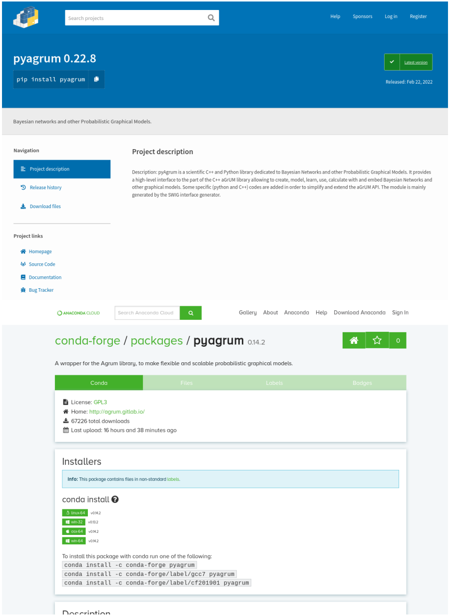
pyAgrum on pipy/anaconda/binder/readthedocs

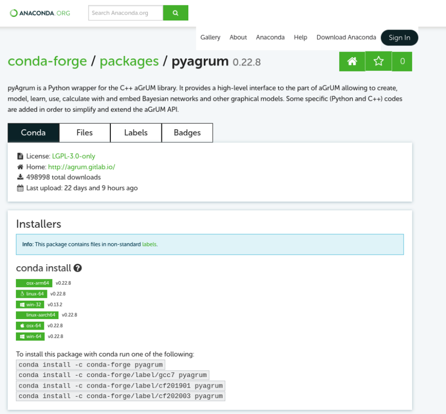




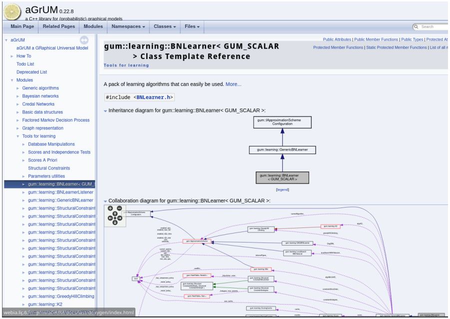


pyAgrum on pipy/anaconda/binder/readthedocs

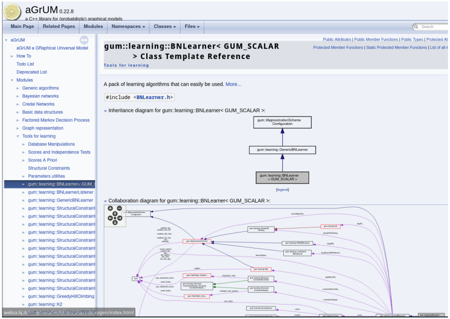


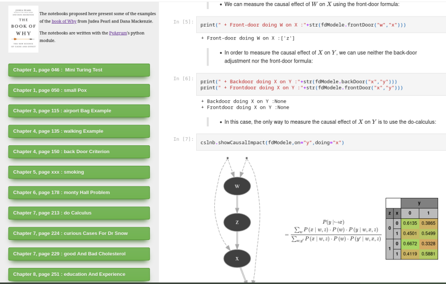


Code quality in aGrUM/pyAgrum : documentation



Code quality in aGrUM/pyAgrum : documentation

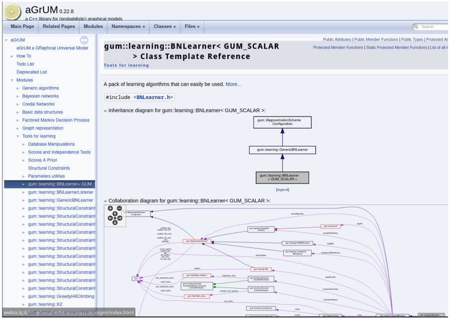


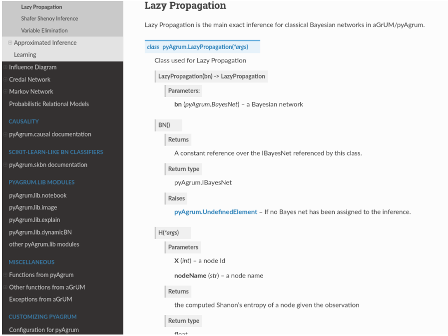


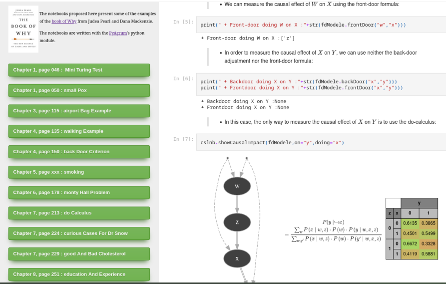
$$P(y \mid \hookrightarrow x) = \frac{\sum_w P(x \mid w,z) \cdot P(w) \cdot P(y \mid w,x,z)}{\sum_{w,y'} P(x \mid w,z) \cdot P(w) \cdot P(y' \mid w,x,z)}$$

z	x	y = 0	y = 1
0	0	0.6135	0.3865
0	1	0.4501	0.5499
1	0	0.6672	0.3328
1	1	0.4119	0.5881

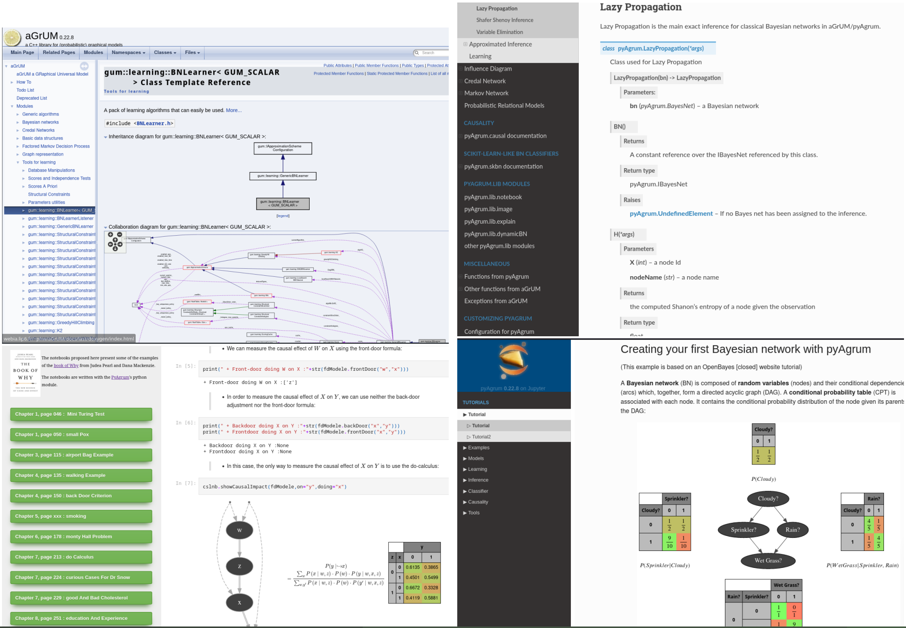
Code quality in aGrUM/pyAgrum : documentation



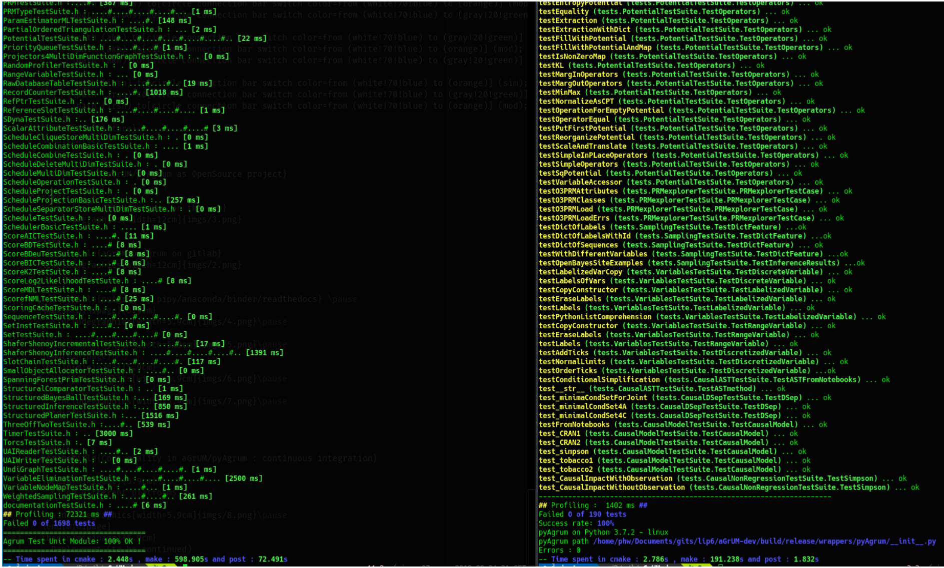




Code quality in aGrUM/pyAgrum : documentation

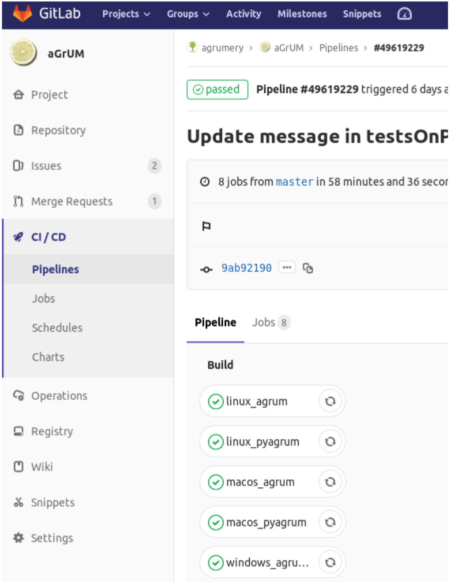


Code quality in aGrUM/pyAgrum : tests

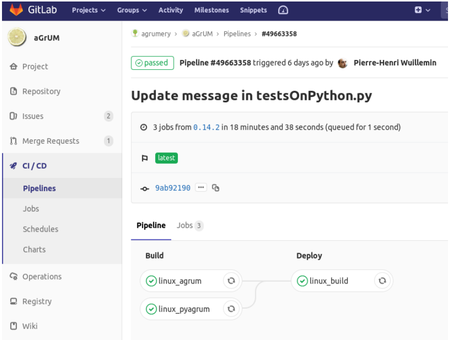


Code quality in aGrUM/pyAgrum : continuous integration

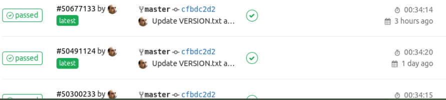
CI on different platforms



Deployment (to be continued)



Nightly build (and tests)



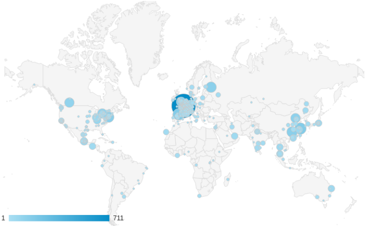
Some stats

Some stats

- Visits (`readthedocs`, `agrum.org`, `notebooks`)

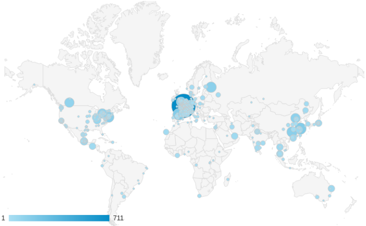
Some stats

- Visits (`readthedocs`, `agrum.org`, `notebooks`)



Some stats

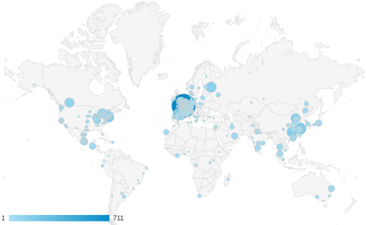
- Visits (`readthedocs`, `agrum.org`, `notebooks`)



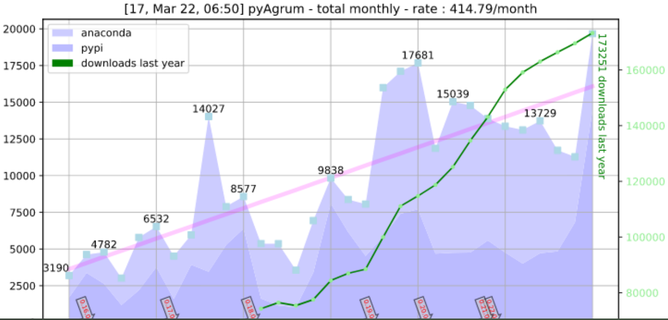
- Téléchargements

Some stats

- Visits (`readthedocs`, `agrum.org`, `notebooks`)



- Téléchargements



And now ?

- aGrUM/pyAgrum still a lab/academic tool. We will not stop maintaining & developing !

And now ?

- aGrUM/pyAgrum still a lab/academic tool. We will not stop maintaining & developing !
- Many users imply many responsabilities

And now ?

- aGrUM/pyAgrum still a lab/academic tool. We will not stop maintaining & developing !
- Many users imply many responsabilities
  - Interaction

And now ?

- aGrUM/pyAgrum still a lab/academic tool. We will not stop maintaining & developing !
- Many users imply many responsabilities
  - Interaction
    gitlab issues, discord, gitter, linkedin, researchGate, what else ?
  - Structuration

And now ?

- aGrUM/pyAgrum still a lab/academic tool. We will not stop maintaining & developing !
- Many users imply many responsabilities
  - Interaction
    gitlab issues, discord, gitter, linkedin, researchGate, what else ?
  - Structuration
    communauty (?), consortium (?)
  - Scientific orientation ?
    - models
    - algorithms
    - scientific committee
    - ?

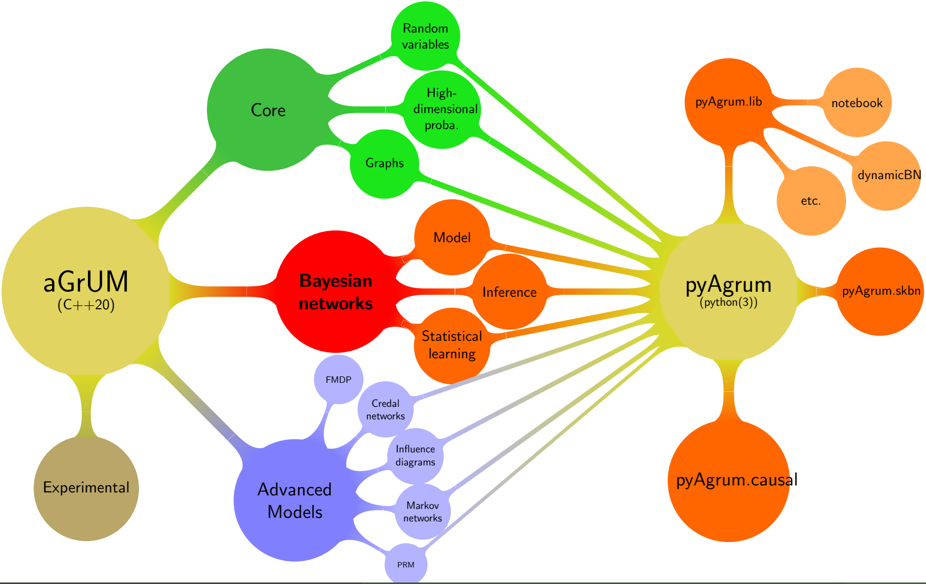
And now ?

- aGrUM/pyAgrum still a lab/academic tool. We will not stop maintaining & developing !
- Many users imply many responsabilities
  - Interaction
    gitlab issues, discord, gitter, linkedin, researchGate, what else ?
  - Structuration
    communauty (?), consortium (?)
  - Scientific orientation ?
    - models
    - algorithms
    - scientific committee
    - ?
  - Development orientation ?

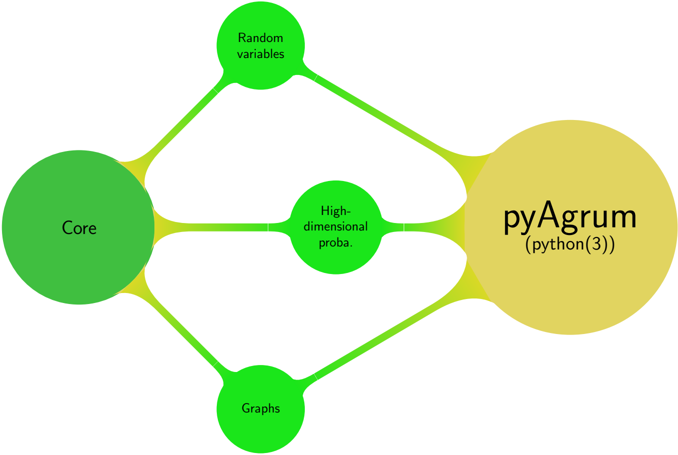
And now ?

- aGrUM/pyAgrum still a lab/academic tool. We will not stop maintaining & developing !
- Many users imply many responsabilities
  - Interaction
    gitlab issues, discord, gitter, linkedin, researchGate, what else ?
  - Structuration
    communauty (?), consortium (?)
  - Scientific orientation ?
    - models
    - algorithms
    - scientific committee
    - ?
  - Development orientation ?
    - weaknesses, strengths
    - missing features
    - Ragrum, JSagrum
    - Steering committee
    - ?

Introspection : focus on 3 elementary components



Introspection : focus on 3 elementary components



Representation of Discrete Variable

Representation of Discrete Variable

Data structure : `DiscreteVariable`

goal : map a finite domain $[0, \cdots, \mathit{domainSize}]$ on a list of labels.

Representation of Discrete Variable

Data structure : `DiscreteVariable`

goal : map a finite domain $[0, \cdots, \textit{domainSize}]$ on a list of labels.

For a `DiscreteVariable` `X` that can take the values a, e, i, o, u, y, `X` is represented by an array :

index	**0**	**1**	**2**	**3**	**4**	**5**
label	a	e	i	o	u	y

Representation of Discrete Variable

Data structure : `DiscreteVariable`

goal : map a finite domain $[0, \cdots, domainSize]$ on a list of labels.

For a `DiscreteVariable` `X` that can take the values a, e, i, o, u, y, `X` is represented by an array :

index	**0**	**1**	**2**	**3**	**4**	**5**
label	a	e	i	o	u	y

The kind of labels defines 4 different types of discrete variables :

Representation of Discrete Variable

Data structure : `DiscreteVariable`

goal : map a finite domain $[0, \cdots, domainSize]$ on a list of labels.

For a `DiscreteVariable` `X` that can take the values a, e, i, o, u, y, `X` is represented by an array :

index	**0**	**1**	**2**	**3**	**4**	**5**
label	a	e	i	o	u	y

The kind of labels defines 4 different types of discrete variables :

- `LabelizedVariable` : list of generic labels (as string),

Representation of Discrete Variable

Data structure : DiscreteVariable

goal : map a finite domain $[0, \cdots, \mathit{domainSize}]$ on a list of labels.

For a DiscreteVariable X that can take the values a, e, i, o, u, y, X is represented by an array :

index	0	1	2	3	4	5
label	a	e	i	o	u	y

The kind of labels defines 4 different types of discrete variables :

- LabelizedVariable : list of generic labels (as string),
- RangeVariable : list of continguous integer labels,

Representation of Discrete Variable

Data structure : `DiscreteVariable`

goal : map a finite domain $[0, \cdots, domainSize]$ on a list of labels.

For a `DiscreteVariable X` that can take the values a, e, i, o, u, y, `X` is represented by an array :

index	**0**	**1**	**2**	**3**	**4**	**5**
label	a	e	i	o	u	y

The kind of labels defines 4 different types of discrete variables :

- `LabelizedVariable` : list of generic labels (as string),
- `RangeVariable` : list of continguous integer labels,
- `DiscretizedVariable` : list of labels defined by a list of float ticks (see below),

Representation of Discrete Variable

Data structure : `DiscreteVariable`

goal : map a finite domain $[0, \cdots, domainSize]$ on a list of labels.

For a `DiscreteVariable` `X` that can take the values a, e, i, o, u, y, `X` is represented by an array :

index	**0**	**1**	**2**	**3**	**4**	**5**
label	a	e	i	o	u	y

The kind of labels defines 4 different types of discrete variables :

- `LabelizedVariable` : list of generic labels (as string),
- `RangeVariable` : list of continguous integer labels,
- `DiscretizedVariable` : list of labels defined by a list of float ticks (see below),
- `IntegerVariable` : list of non-contiguous integer labels.

Representation of Discrete Variable

Data structure : `DiscreteVariable`

goal : map a finite domain $[0, \cdots, domainSize]$ on a list of labels.

For a `DiscreteVariable` `X` that can take the values a, e, i, o, u, y, `X` is represented by an array :

index	**0**	**1**	**2**	**3**	**4**	**5**
label	a	e	i	o	u	y

The kind of labels defines 4 different types of discrete variables :

- `LabelizedVariable` : list of generic labels (as string),
- `RangeVariable` : list of continguous integer labels,
- `DiscretizedVariable` : list of labels defined by a list of float ticks (see below),
- `IntegerVariable` : list of non-contiguous integer labels.

Discrete variables as list of labels

The kind of labels defines 4 different types of discrete variables :

- `LabelizedVariable` : list of generic labels (as string),
- `RangeVariable` : list of continguous integer labels,
- `DiscretizedVariable` : list of labels defined by a list of float ticks (see below),
- `IntegerVariable` : list of non-contiguous integer labels.

Discrete variables as list of labels

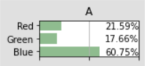
The kind of labels defines 4 different types of discrete variables :

- `LabelizedVariable` : list of generic labels (as string),
- `RangeVariable` : list of continguous integer labels,
- `DiscretizedVariable` : list of labels defined by a list of float ticks (see below),
- `IntegerVariable` : list of non-contiguous integer labels.

gum.LabelizedVariable

`A{Red|Green|Blue}`

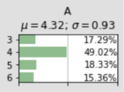
A		
Red	Green	Blue
0.2159	0.1766	0.6075



gum.RangeVariable

`A[3,6]`

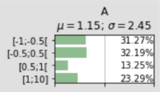
A			
3	4	5	6
0.1729	0.4902	0.1833	0.1536



gum.DiscretizedVariable

`A[-1,-0.5,0.5,1,10]`

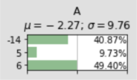
A			
[-1;-0.5[	[-0.5;0.5[	[0.5;1[	[1;10]
0.3127	0.3219	0.1325	0.2329



gum.IntegerVariable

`A{-14|5|6}`

A		
-14	5	6
0.4087	0.0973	0.4940



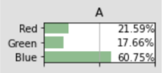
```python
def aff(name,fastStx):
    bn=gum.fastBN(fastStx)
    return (f"<h2>{name}</h2>",
            f"<tt>{fastStx}</tt>",
            gnb.getPotential(bn.cpt(0)),
            gnb.getPosterior(bn,target=0,evs={}))

gnb.sideBySide(*aff("gum.LabelizedVariable","A{Red|Green|Blue}"),
               *aff("gum.RangeVariable","A[3,6]"),
               *aff("gum.DiscretizedVariable","A[-1,-0.5,0.5,1,10]"),
               *aff("gum.IntegerVariable","A{-14|5|6}"),
              ncols=4)
```

gum.LabelizedVariable

`A{Red|Green|Blue}`

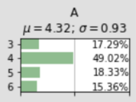
A		
Red	Green	Blue
0.2159	0.1766	0.6075



gum.RangeVariable

`A[3,6]`

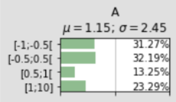
A			
3	4	5	6
0.1729	0.4902	0.1833	0.1536



gum.DiscretizedVariable

`A[-1,-0.5,0.5,1,10]`

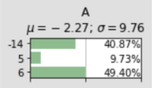
A			
[-1;-0.5[	[-0.5;0.5[	[0.5;1[	[1;10]
0.3127	0.3219	0.1325	0.2329



gum.IntegerVariable

`A{-14|5|6}`

A		
-14	5	6
0.4087	0.0973	0.4940



A shared and simple API for all Discrete Variables

A shared and simple API for all Discrete Variables

```python
def apiDiscreteVar(variable,value,position):
    print(f"{variable} : {value=}, {position=}")
    print(f" + {variable.domainSize()=}")
    print(f" + {variable[value]=}")
    print(f" + {variable.index(value)=}")
    print(f" + {variable.label(position)=}")

apiDiscreteVar(bn.variable("A"),"Green",0)
apiDiscreteVar(bn.variable("B"),"5",0)
apiDiscreteVar(bn.variable("C"),"0.75",0)
apiDiscreteVar(bn.variable("D"),"5",0)
```

```
A:Labelized(<Red,Green,Blue>) : value='Green', position=0
 + variable.domainSize()=3
 + variable[value]=1
 + variable.index(value)=1
 + variable.label(position)='Red'
B:Range([3,6]) : value='5', position=0
 + variable.domainSize()=4
 + variable[value]=2
 + variable.index(value)=2
 + variable.label(position)='3'
C:Discretized(<[-1;-0.5[,[-0.5;0.5[,[0.5;1[,[1;10]>) : value='0.75', position=0
 + variable.domainSize()=4
 + variable[value]=2
 + variable.index(value)=2
 + variable.label(position)='[-1;-0.5['
D:Integer(<-14,5,6>) : value='5', position=0
 + variable.domainSize()=3
 + variable[value]=1
 + variable.index(value)=1
 + variable.label(position)='-14'
```

Representation of graphs

Data structure : `Directed|Mixed|UnorientedGraph`

goal : represent a list of `Arc` (a, b) and `Edge` $\{b, a\}$ between positive integers.

Representation of graphs

Data structure : `Directed|Mixed|UnorientedGraph`

goal : represent a list of `Arc` (a, b) and `Edge` $\{b, a\}$ between positive integers.

- Very compact definition of a graph : not even explicit set of nodes

Representation of graphs

Data structure : `Directed|Mixed|UnorientedGraph`

goal : represent a list of `Arc` (a, b) and `Edge` $\{b, a\}$ between positive integers.

- Very compact definition of a graph : not even explicit set of nodes
- Nodes, edges and arcs will be annotated for more complex structures based on graphs.

Representation of graphs

Data structure : `Directed|Mixed|UnorientedGraph`

goal : represent a list of `Arc` (a, b) and `Edge` $\{b, a\}$ between positive integers.

- Very compact definition of a graph : not even explicit set of nodes
- Nodes, edges and arcs will be annotated for more complex structures based on graphs.
- the nodes (`unsigned long`) are called `NodeId`

Representation of graphs

Data structure : `Directed|Mixed|UnorientedGraph`

goal : represent a list of `Arc` (a, b) and `Edge` $\{b, a\}$ between positive integers.

- Very compact definition of a graph : not even explicit set of nodes
- Nodes, edges and arcs will be annotated for more complex structures based on graphs.
- the nodes (`unsigned long`) are called `NodeId`

Several type of graphs :

- `DiGraph`
  - `DAG (Directed Acyclic Graph)`
- `UndiGraph` (and `CliqueGraph`)
- `MixedGraph`

API for graphs

- node : `addNode`, `addNodeWithId(a)`, `addNodes(nbr)` , `eraseNode(a)`

API for graphs

- node : `addNode`, `addNodeWithId(a)`, `addNodes(nbr)` , `eraseNode(a)`
- arcs, edges : `addEdge`, `eraseEdge`, etc.

API for graphs

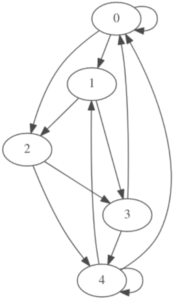
- node : addNode, addNodeWithId(a), addNodes(nbr) , eraseNode(a)
- arcs, edges : addEdge, eraseEdge, etc.
- accessors : parents,children,neighbour, etc.
- algorithms : topologicalOrder, moralGraph, connectedComponents, etc.
- visualisation : toDot() (used by pyAgrum.lib.notebook for instance)

Digraph

```
g=gum.DiGraph()
g.addNodes(5) # returns the generated nodeId
```

```
{0, 1, 2, 3, 4}
```

```
for i in range(5):
    g.addArc(i,(i+1)%5)
    g.addArc(i,(i+2)%5)
g.addArc(0,0)
g.addArc(4,4)
g
```



API for graphs

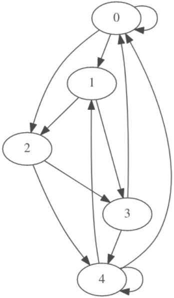
- node : `addNode`, `addNodeWithId(a)`, `addNodes(nbr)` , `eraseNode(a)`
- arcs, edges : `addEdge`, `eraseEdge`, etc.
- accessors : `parents`,`children`,`neighbour`, etc.
- algorithms : `topologicalOrder`, `moralGraph`, `connectedComponents`, etc.
- visualisation : `toDot() (used by pyAgrum.lib.notebook for instance)`

Digraph

```python
g=gum.DiGraph()
g.addNodes(5) # returns the generated nodeId
```

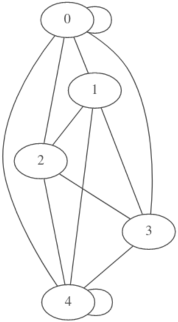
```
{0, 1, 2, 3, 4}
```

```python
for i in range(5):
    g.addArc(i,(i+1)%5)
    g.addArc(i,(i+2)%5)
g.addArc(0,0)
g.addArc(4,4)
g
```



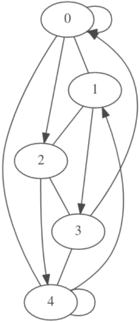
UndiGraph

```python
g=gum.UndiGraph()
g.addNodes(5)
for i in range(5):
    g.addEdge(i,(i+1)%5)
    g.addEdge(i,(i+2)%5)
g.addEdge(0,0)
g.addEdge(4,4)
g
```



Mixed Graph

```python
g=gum.MixedGraph()
g.addNodes(5)
for i in range(5):
    g.addEdge(i,(i+1)%5)
    g.addArc(i,(i+2)%5)
g.addArc(0,0)
g.addEdge(4,4)
g
```



Representation of multi-dimensionnal arrays

Representation of multi-dimensionnal arrays

Data structure : `Potential`

goal : representation of multi-dimensional arrays (of float) without ambiguity on dimensions.

Representation of multi-dimensionnal arrays

Data structure : `Potential`

goal : representation of multi-dimensional arrays (of float) without ambiguity on dimensions.

Implementing tensor algebra

$$f = g + h$$

Representation of multi-dimensionnal arrays

Data structure : `Potential`

goal : representation of multi-dimensional arrays (of float) without ambiguity on dimensions.

Implementing tensor algebra

$$f(,,) = g(,,) + h(,)$$

Representation of multi-dimensionnal arrays

Data structure : `Potential`

goal : representation of multi-dimensional arrays (of float) without ambiguity on dimensions.

Implementing tensor algebra

$$f(,,) = g(,,) + h(,)$$

[[[1. 2.] [3. 4.]] [[5. 6.] [7. 8.]]]	[[1. 3.] [5. 7.]]	?
$g(,,)$	$h(,)$	$f(,,) = g(,,) + h(,)$

Representation of multi-dimensionnal arrays

Data structure : `Potential`

goal : representation of multi-dimensional arrays (of float) without ambiguity on dimensions.

Implementing tensor algebra

$$f(a, b, c) = g(b, a, c) + h(b, c)$$

[[[1. 2.] [3. 4.]] [[5. 6.] [7. 8.]]]	[[1. 3.] [5. 7.]]	?
$g(,,)$	$h(,)$	$f(,,) = g(,,) + h(,)$

Representation of multi-dimensionnal arrays

Data structure : `Potential`

goal : representation of multi-dimensional arrays (of float) without ambiguity on dimensions.

Implementing tensor algebra

$$f(a, b, c) = g(b, a, c) + h(b, c)$$

		A	
C	**B**	**0**	**1**
0	**0**	1.0000	2.0000
	1	3.0000	4.0000
1	**0**	5.0000	6.0000
	1	7.0000	8.0000

$g(a, b, c)$

	B	
C	**0**	**1**
0	1.0000	3.0000
1	5.0000	7.0000

$h(b, c)$

		B	
A	**C**	**0**	**1**
0	**0**	2.0000	6.0000
	1	10.0000	14.0000
1	**0**	3.0000	7.0000
	1	11.0000	15.0000

$f(a, b, c) = g(a, b, c) + h(b, c)$

API for Potential

Quite complex API.

API for Potential

Quite complex API.

Creation and operations on Potential

API for Potential

Quite complex API.

Creation and operations on Potential

```
a=gum.LabelizedVariable("A","descr of A",2)
b=gum.LabelizedVariable("B","descr of B",2)
c=gum.LabelizedVariable("C","descr of C",2)

g=gum.Potential().add(b).add(a).add(c).fillWith([1,2,3,4,5,6,7,8])
h=gum.Potential().add(b).add(c).fillWith([1,3,5,7])

f=g+h
```

API for Potential

Quite complex API.

Creation and operations on Potential

```
a=gum.LabelizedVariable("A","descr of A",2)
b=gum.LabelizedVariable("B","descr of B",2)
c=gum.LabelizedVariable("C","descr of C",2)

g=gum.Potential().add(b).add(a).add(c).fillWith([1,2,3,4,5,6,7,8])
h=gum.Potential().add(b).add(c).fillWith([1,3,5,7])

f=g+h
```

Methods on Potential

API for Potential

Quite complex API.

Creation and operations on Potential

```
a=gum.LabelizedVariable("A","descr of A",2)
b=gum.LabelizedVariable("B","descr of B",2)
c=gum.LabelizedVariable("C","descr of C",2)

g=gum.Potential().add(b).add(a).add(c).fillWith([1,2,3,4,5,6,7,8])
h=gum.Potential().add(b).add(c).fillWith([1,3,5,7])

f=g+h
```

Methods on Potential

```
gnb.sideBySide(f,f.sum(),f.margSumOut("B"),
              captions=['$$f(a,b,c)$$','$$\sum_{a,b,c} f(a,b,c)$$','$$\sum_{b} f(a,b,c)$$'])
```

		B	
A	C	0	1
0	0	2.0000	5.0000
	1	10.0000	13.0000
1	0	4.0000	7.0000
	1	12.0000	15.0000

$f(a,b,c)$

68.0

$\sum_{a,b,c} f(a,b,c)$

	C	
A	0	1
0	7.0000	23.0000
1	11.0000	27.0000

$\sum_{b} f(a,b,c)$

API for Potential

Quite complex API.

Creation and operations on Potential

```
a=gum.LabelizedVariable("A","descr of A",2)
b=gum.LabelizedVariable("B","descr of B",2)
c=gum.LabelizedVariable("C","descr of C",2)

g=gum.Potential().add(b).add(a).add(c).fillWith([1,2,3,4,5,6,7,8])
h=gum.Potential().add(b).add(c).fillWith([1,3,5,7])

f=g+h
```

Methods on Potential

```
gnb.sideBySide(f,f.sum(),f.margSumOut("B"),
               captions=['$$f(a,b,c)$$','$$\sum_{a,b,c} f(a,b,c)$$','$$\sum_{b} f(a,b,c)$$'])
```

		B	
A	C	0	1
0	0	2.0000	5.0000
	1	10.0000	13.0000
1	0	4.0000	7.0000
	1	12.0000	15.0000

$f(a,b,c)$

68.0

$\sum_{a,b,c} f(a,b,c)$

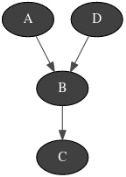
	C	
A	0	1
0	7.0000	23.0000
1	11.0000	27.0000

$\sum_{b} f(a,b,c)$

Potential for probabilities

Potential for probabilities

```
bn=gum.fastBN("A->B->C;D->B")
gnb.sideBySide(bn,bn.cpt("A"),bn.cpt("B"),bn.cpt("C"),bn.cpt("D"))
```



A	
0	1
0.1495	0.8505

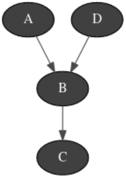
		B	
D	A	0	1
0	0	0.6356	0.3644
	1	0.6883	0.3117
1	0	0.2372	0.7628
	1	0.3500	0.6500

	C	
B	0	1
0	0.5294	0.4706
1	0.3654	0.6346

D	
0	1
0.7158	0.2842

Potential for probabilities

```
bn=gum.fastBN("A->B->C;D->B")
gnb.sideBySide(bn,bn.cpt("A"),bn.cpt("B"),bn.cpt("C"),bn.cpt("D"))
```



A	
0	1
0.1495	0.8505

		B	
D	A	0	1
0	0	0.6356	0.3644
	1	0.6883	0.3117
1	0	0.2372	0.7628
	1	0.3500	0.6500

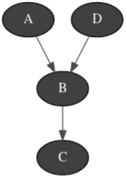
	C	
B	0	1
0	0.5294	0.4706
1	0.3654	0.6346

D	
0	1
0.7158	0.2842

$$P(A, B, C, D) = P(A) * P(D) * P(B|A, D) * P(C|B)$$

Potential for probabilities

```
bn=gum.fastBN("A->B->C;D->B")
gnb.sideBySide(bn,bn.cpt("A"),bn.cpt("B"),bn.cpt("C"),bn.cpt("D"))
```



A	
0	1
0.1495	0.8505

D	A	B: 0	B: 1
0	0	0.6356	0.3644
0	1	0.6883	0.3117
1	0	0.2372	0.7628
1	1	0.3500	0.6500

B	C: 0	C: 1
0	0.5294	0.4706
1	0.3654	0.6346

D	
0	1
0.7158	0.2842

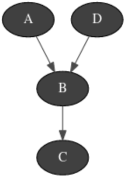
$$P(A, B, C, D) = P(A) * P(D) * P(B|A, D) * P(C|B)$$

```
pABCD=bn.cpt("A")*bn.cpt("B")*bn.cpt("C")*bn.cpt("D")
pABCD
```

A	C	B	D: 0	D: 1
0	0	0	0.0360	0.0053
0	0	1	0.0143	0.0118
0	1	0	0.0320	0.0047
0	1	1	0.0247	0.0206
1	0	0	0.2218	0.0448
1	0	1	0.0693	0.0574
1	1	0	0.1972	0.0398
1	1	1	0.1204	0.0997

Potential for probabilities

```
bn=gum.fastBN("A->B->C;D->B")
gnb.sideBySide(bn,bn.cpt("A"),bn.cpt("B"),bn.cpt("C"),bn.cpt("D"))
```



A	
0	1
0.1495	0.8505

		B	
D	A	0	1
0	0	0.6356	0.3644
	1	0.6883	0.3117
1	0	0.2372	0.7628
	1	0.3500	0.6500

	C	
B	0	1
0	0.5294	0.4706
1	0.3654	0.6346

D	
0	1
0.7158	0.2842

$$P(A, B, C, D) = P(A) * P(D) * P(B|A, D) * P(C|B)$$

```
pABCD=bn.cpt("A")*bn.cpt("B")*bn.cpt("C")*bn.cpt("D")
pABCD
```

			D	
A	C	B	0	1
0	0	0	0.0360	0.0053
		1	0.0143	0.0118
	1	0	0.0320	0.0047
		1	0.0247	0.0206
1	0	0	0.2218	0.0448
		1	0.0693	0.0574
	1	0	0.1972	0.0398
		1	0.1204	0.0997

Potential for probabilities (2)

Potential for probabilities (2)

$P(D|C)$

Potential for probabilities (2)

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

Potential for probabilities (2)

$$P(D|C) = \frac{P(D,C)}{P(C)} = \frac{\sum_{A,B} P(A,B,C,D)}{\sum_{A,B,D} P(A,B,C,D)}$$

Potential for probabilities (2)

$$P(D|C) = \frac{P(D, C)}{P(C)} = \frac{\sum_{A,B} P(A, B, C, D)}{\sum_{A,B,D} P(A, B, C, D)}$$

```
pABCD.margSumOut(["A","B"])/pABCD.margSumIn("C")
```

	C	
D	0	1
0	0.7409	0.6943
1	0.2591	0.3057

Potential for probabilities (2)

$$P(D|C) = \frac{P(D,C)}{P(C)} = \frac{\sum_{A,B} P(A,B,C,D)}{\sum_{A,B,D} P(A,B,C,D)}$$

```
pABCD.margSumOut(["A","B"])/pABCD.margSumIn("C")
```

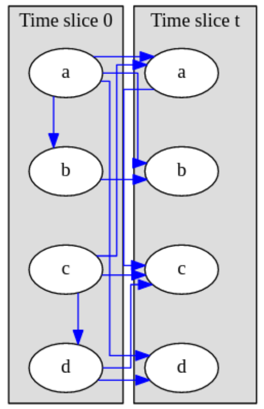
	C	
D	0	1
0	0.7409	0.6943
1	0.2591	0.3057

Using these components to build new model in pyAgrum

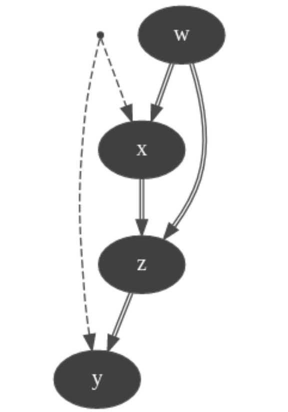
Using these components to build new model in pyAgrum

Two models exist only in pyAgrum and has been developped mainly from those components :

dynamic Bayesian Network



Causal model



dynamic Bayesian Networks

dBN (dynamic BN)

dynamic Bayesian Networks

dBN (dynamic BN)

a dynamic Bayesian network is a Bayesian network with wariables indexed by the time t and by i : $\mathbf{X}^{(\mathbf{t})} = X_1^{(t)}, \cdots, X_N^{(t)}$

dynamic Bayesian Networks

dBN (dynamic BN)

a dynamic Bayesian network is a Bayesian network with wariables indexed by the time t and by i : $\mathbf{X}^{(\mathbf{t})} = X_1^{(t)}, \cdots, X_N^{(t)}$ for which those properties hold :

dynamic Bayesian Networks

dBN (dynamic BN)

a dynamic Bayesian network is a Bayesian network with wariables indexed by the time t and by i : $\mathbf{X}^{(\mathbf{t})} = X_1^{(t)}, \cdots, X_N^{(t)}$ for which those properties hold :

- Markov property :

dynamic Bayesian Networks

dBN (dynamic BN)

a dynamic Bayesian network is a Bayesian network with wariables indexed by the time t and by i : $\mathbf{X^{(t)}} = X_1^{(t)}, \cdots, X_N^{(t)}$ for which those properties hold :

- Markov property :
  $P(\mathbf{X^{(t)}} \mid \mathbf{X^{(0)}}, \cdots, \mathbf{X^{(t-1)}}) = P(\mathbf{X^{(t)}} \mid \mathbf{X^{(t-1)}})$,
- Homogeneity :

dynamic Bayesian Networks

dBN (dynamic BN)

a dynamic Bayesian network is a Bayesian network with wariables indexed by the time t and by i : $\mathbf{X^{(t)}} = X_1^{(t)}, \cdots, X_N^{(t)}$ for which those properties hold :

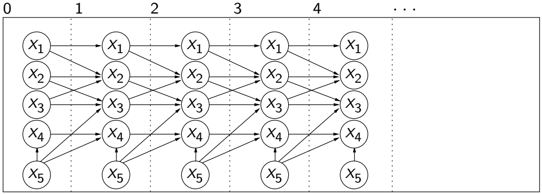
- Markov property :
  $P(\mathbf{X^{(t)}} \mid \mathbf{X^{(0)}}, \cdots, \mathbf{X^{(t-1)}}) = P(\mathbf{X^{(t)}} \mid \mathbf{X^{(t-1)}})$,
- Homogeneity :
  $P(\mathbf{X^{(t)}} \mid \mathbf{X^{(t-1)}}) = \cdots = P(\mathbf{X^{(1)}} \mid \mathbf{X^{(0)}}))$.

dynamic Bayesian Networks

dBN (dynamic BN)

a dynamic Bayesian network is a Bayesian network with wariables indexed by the time t and by i : $\mathbf{X^{(t)}} = X_1^{(t)}, \cdots, X_N^{(t)}$ for which those properties hold :

- Markov property :
  $P(\mathbf{X^{(t)}} \mid \mathbf{X^{(0)}}, \cdots, \mathbf{X^{(t-1)}}) = P(\mathbf{X^{(t)}} \mid \mathbf{X^{(t-1)}})$,
- Homogeneity :
  $P(\mathbf{X^{(t)}} \mid \mathbf{X^{(t-1)}}) = \cdots = P(\mathbf{X^{(1)}} \mid \mathbf{X^{(0)}}))$.



dynamic Bayesian networks : 2-TBN

2-TBN

A dynamic Bayesian network is defined by

dynamic Bayesian networks : 2-TBN

2-TBN

A dynamic Bayesian network is defined by

- initial distributions ($P(X^{(0)})$),

dynamic Bayesian networks : 2-TBN

2-TBN

A dynamic Bayesian network is defined by

- initial distributions ($P(X^{(0)})$,
- the transition between the variables at time $t-1$ and the same varaibles at time t (*timeslices*).

dynamic Bayesian networks : 2-TBN

2-TBN

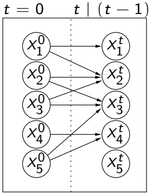
A dynamic Bayesian network is defined by

- initial distributions ($P(X^{(0)})$,
- the transition between the variables at time $t-1$ and the same varaibles at time t (*timeslices*).

2-TBN

A dynamic Bayesian network is defined by

- initial distributions ($P(X^{(0)})$,
- the transition between the variables at time $t-1$ and the same varaibles at time t (*timeslices*).



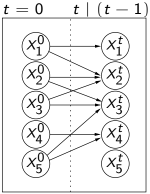
$P(x_1^{(t)}, \ldots, x_5^{(t)} \mid x_1^{(t-1)}, \ldots, x_5^{(t-1)})$

dynamic Bayesian networks : 2-TBN

2-TBN

A dynamic Bayesian network is defined by

- initial distributions ($P(X^{(0)})$,
- the transition between the variables at time $t-1$ and the same varaibles at time t (*timeslices*).



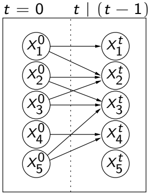
$$P(x_1^{(t)}, \dots, x_5^{(t)} \mid x_1^{(t-1)}, \dots, x_5^{(t-1)}) \quad = P(x_1^{(t)} \mid x^{(t-1)})$$

dynamic Bayesian networks : 2-TBN

2-TBN

A dynamic Bayesian network is defined by

- initial distributions ($P(X^{(0)})$,
- the transition between the variables at time $t-1$ and the same varaibles at time t (*timeslices*).

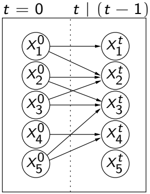


$$P(x_1^{(t)},\ldots,x_5^{(t)} \mid x_1^{(t-1)},\ldots,x_5^{(t-1)}) \quad = P(x_1^{(t)} \mid x^{(t-1)})\, P(x_2^{(t)} \mid x_1^{(t-1)}, x_2^{(t-1)}, x_3^{(t-1)})$$

2-TBN

A dynamic Bayesian network is defined by

- initial distributions ($P(X^{(0)})$,
- the transition between the variables at time $t-1$ and the same varaibles at time t (*timeslices*).



$$
\begin{aligned}
P(x_1^{(t)}, \dots, x_5^{(t)} \mid x_1^{(t-1)}, \dots, x_5^{(t-1)}) \quad &= P(x_1^{(t)} \mid x^{(t-1)}) \\
&P(x_2^{(t)} \mid x_1^{(t-1)}, x_2^{(t-1)}, x_3^{(t-1)}) \\
&P(x_3^{(t)} \mid x_2^{(t-1)}, x_3^{(t-1)}, x_4^{(t-1)}) \\
&P(x_4^{(t)} \mid x_4^{(t-1)}, x_5^{(t-1)})
\end{aligned}
$$

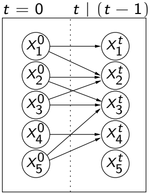
1024 versus 4+16+16+8+2=46 ! !

dynamic Bayesian networks : 2-TBN

2-TBN

A dynamic Bayesian network is defined by

- initial distributions $(P(X^{(0)})$,
- the transition between the variables at time $t-1$ and the same varaibles at time t (*timeslices*).



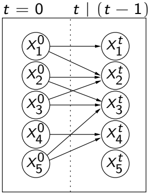
$$P(x_1^{(t)},\ldots,x_5^{(t)} \mid x_1^{(t-1)},\ldots,x_5^{(t-1)}) \quad = P(x_1^{(t)} \mid x^{(t-1)})\, P(x_2^{(t)} \mid x_1^{(t-1)}, x_2^{(t-1)}, x_3^{(t-1)})\, P(x_3^{(t)} \mid x_2^{(t-1)}, x_3^{(t-1)}, x_4^{(t-1)})\, P(x_4^{(t)} \mid x_4^{(t-1)}, x_5^{(t-1)})\, P(x_5^{(t)})$$

1024 versus 4+16+16+8+2=46 ! !

2-TBN

A dynamic Bayesian network is defined by

- initial distributions ($P(X^{(0)})$),
- the transition between the variables at time $t-1$ and the same varaibles at time t (*timeslices*).



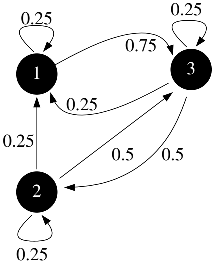
$$P(x_1^{(t)},\dots,x_5^{(t)} \mid x_1^{(t-1)},\dots,x_5^{(t-1)})$$

$$\begin{aligned} = & P(x_1^{(t)} \mid x^{(t-1)}) \\ & P(x_2^{(t)} \mid x_1^{(t-1)}, x_2^{(t-1)}, x_3^{(t-1)}) \\ & P(x_3^{(t)} \mid x_2^{(t-1)}, x_3^{(t-1)}, x_4^{(t-1)}) \\ & P(x_4^{(t)} \mid x_4^{(t-1)}, x_5^{(t-1)}) \\ & P(x_5^{(t)}) \end{aligned}$$

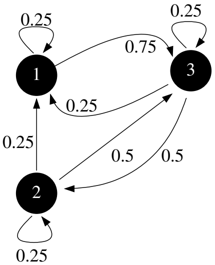
1024 versus 4+16+16+8+2=46 ! !

The representation a.k.a 2TBN (2 timeslices BN) allow the modelisation of a virtually infinite Bayesian network which is the unrolled model from time 0.

Markov Chain and dynamic Bayesian network



Markov Chain and dynamic Bayesian network



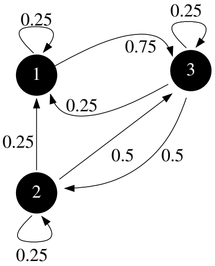
$$P(X^n \mid X^{n-1}) = \begin{pmatrix} 0.25 & 0 & 0.75 \\ 0.25 & 0.25 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{pmatrix}$$

Markov chain

- a discrete variable (X^n) (at time n).
- Parameters for this model :
  - Initial condition : $P(X^O)$
  - transition probabilities : $P(X^n \mid X^{n-1})$

Equivalent dynamic Bayesian network :

Markov Chain and dynamic Bayesian network



$$P(X^n \mid X^{n-1}) = \begin{pmatrix} 0.25 & 0 & 0.75 \\ 0.25 & 0.25 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{pmatrix}$$

Markov chain

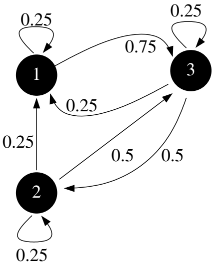
- a discrete variable (X^n) (at time n).
- Parameters for this model :
  - Initial condition : $P(X^O)$
  - transition probabilities : $P(X^n \mid X^{n-1})$

Equivalent dynamic Bayesian network :

dBN : $X^0 \longrightarrow X^1 \longrightarrow X^2 \longrightarrow X^3 \longrightarrow \cdots$

2TBN : $\boxed{X^0} \longrightarrow \boxed{X^n}$

Markov Chain and dynamic Bayesian network

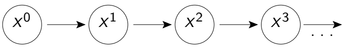


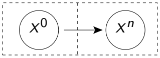
$$P(X^n \mid X^{n-1}) = \begin{pmatrix} 0.25 & 0 & 0.75 \\ 0.25 & 0.25 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{pmatrix}$$

Markov chain

- a discrete variable (X^n) (at time n).
- Parameters for this model :
  - Initial condition : $P(X^O)$
  - transition probabilities : $P(X^n \mid X^{n-1})$

Equivalent dynamic Bayesian network :

dBN : 

2TBN : 

Could we do the same with continuous time ?

Continuous-Time Markov Process

Dynamic processus dynamique verifying :

- a discrete variable

Continuous-Time Markov Process

Dynamic processus dynamique verifying :

- a discrete variable
- a transition from a state to another can happen any time,

$$\underbrace{0 \qquad\qquad}_{X(0)} t_1 \underbrace{\qquad\qquad\qquad\qquad\qquad}_{X(t_1)} t_2 \underbrace{\qquad}_{X(t_2)} t_3 \cdots$$

Continuous-Time Markov Process

Dynamic processus dynamique verifying :

- a discrete variable
- a transition from a state to another can happen any time,

$$\underbrace{0 \qquad\qquad}_{X(0)} t_1 \underbrace{\qquad\qquad\qquad\qquad}_{X(t_1)} t_2 \underbrace{\qquad}_{X(t_2)} t_3 \cdots$$

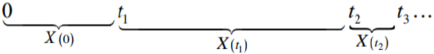
- Continuous time Markov property :

$$\forall s > r, \forall t > 0, P(X(s+t)|X(s)), X(r)) = P(X(s+t)|X(s)))$$

Continuous-Time Markov Process

Dynamic processus dynamique verifying :

- a discrete variable
- a transition from a state to another can happen any time,



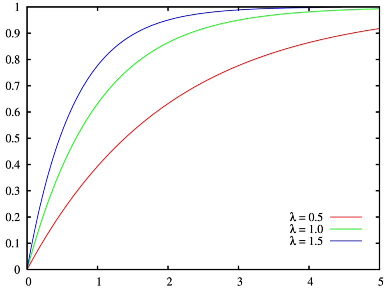
- Continuous time Markov property :

$$\forall s > r, \forall t > 0, P(X(s+t)|X(s)), X(r)) = P(X(s+t)|X(s)))$$

Exponential distribution

$$D \sim Exp(\lambda)$$

- Cumulative distribution function
  $\forall d > 0, F(d) = 1 - e^{-\lambda d}$
- $\mathbb{E}(D) = \lambda^{-1}$
- $\sigma(D) = \lambda^{-1}$



Continuous Time Markov Chain (CTMC)

Continuous Time Markov Chain (CTMC)

A CTMC is a continuous stochastic process in which, for each state, the process will change state according to an exponential random variable and then move to a different state as specified by the probabilities of a stochastic matrix.

Continuous Time Markov Chain (CTMC)

A CTMC is a continuous stochastic process in which, for each state, the process will change state according to an exponential random variable and then move to a different state as specified by the probabilities of a stochastic matrix.

Minimum of independent exponential distributions

Continuous Time Markov Chain (CTMC)

A CTMC is a continuous stochastic process in which, for each state, the process will change state according to an exponential random variable and then move to a different state as specified by the probabilities of a stochastic matrix.

Minimum of independent exponential distributions

$$X \sim Exp(\lambda), Y \sim Exp(\mu), X \perp\!\!\!\perp Y \Rightarrow min(X, Y) \sim Exp(\lambda + \mu)$$

Continuous Time Markov Chain (CTMC)

A CTMC is a continuous stochastic process in which, for each state, the process will change state according to an exponential random variable and then move to a different state as specified by the probabilities of a stochastic matrix.

Minimum of independent exponential distributions

$$X \sim Exp(\lambda), Y \sim Exp(\mu), X \perp\!\!\!\perp Y \Rightarrow min(X, Y) \sim Exp(\lambda + \mu)$$

Continuous Time Markov Chain

$(X_t \in x_1, \cdots, x_n)_{t \geq 0}$ CdMTC is defined by

Continuous Time Markov Chain (CTMC)

A CTMC is a continuous stochastic process in which, for each state, the process will change state according to an exponential random variable and then move to a different state as specified by the probabilities of a stochastic matrix.

Minimum of independent exponential distributions

$$X \sim Exp(\lambda), Y \sim Exp(\mu), X \perp\!\!\!\perp Y \Rightarrow min(X, Y) \sim Exp(\lambda + \mu)$$

Continuous Time Markov Chain

$(X_t \in x_1, \cdots, x_n)_{t \geq 0}$ CdMTC is defined by

- $P(X_0)$
- $\forall i, j \geq n, q_{i,j}$ tels que
    1. $q_{i,i} \in \mathbb{R}^-$
    2. $\forall i \neq j, q_{i,j} \in \mathbb{R}^+$
    3. $\forall i, \sum_j q_{i,j} = 0$

$$Q_X = \begin{pmatrix} -0.21 & 0.20 & 0.01 \\ 0.05 & -0.10 & 0.05 \\ 0.01 & 0.20 & -0.21 \end{pmatrix}$$

intensity matrix

$q_{i,j}$ is the parameter of the exponential distribution controlling the transition from state i to state j
$-q_{i,i}$ is the paramter of the exponential distribution controlling a transition from state i.

Properties

$(X_t \in x_1, \cdots, x_n)_{t \geq 0}$ CTMC :

- $P(X_0)$
- $Q_X = (q_{i,j})_{i \leq n, j \leq n}$ intensity matrix

Properties

$(X_t \in x_1, \cdots, x_n)_{t \geq 0}$ CTMC :

- $P(X_0)$
- $Q_X = (q_{i,j})_{i \leq n, j \leq n}$ intensity matrix

Properties

- $p_{i,j} = \frac{q_{i,j}}{-q_{i,i}}$ is the probability of transition from x_i to x_j

Properties

$(X_t \in x_1, \cdots, x_n)_{t \geq 0}$ CTMC :

- $P(X_0)$
- $Q_X = (q_{i,j})_{i \leq n, j \leq n}$ intensity matrix

Properties

- $p_{i,j} = \frac{q_{i,j}}{-q_{i,i}}$ is the probability of transition from x_i to x_j
- $P(X_t) = P(X_O) \cdot exp(Q_X t)$ with $exp(M) = \sum_{n=0}^{\infty} \frac{M^n}{n!}$

Properties

$(X_t \in x_1, \cdots, x_n)_{t \geq 0}$ CTMC :

- $P(X_0)$
- $Q_X = (q_{i,j})_{i \leq n, j \leq n}$ intensity matrix

Properties

- $p_{i,j} = \frac{q_{i,j}}{-q_{i,i}}$ is the probability of transition from x_i to x_j
- $P(X_t) = P(X_O) \cdot exp(Q_X t)$ with $exp(M) = \sum_{n=0}^{\infty} \frac{M^n}{n!}$

Convergence

With some good conditions (ergodicity), $P(X_t) \underset{t \to \infty}{\longrightarrow} P(X^*)$

Properties

$(X_t \in x_1, \cdots, x_n)_{t \geq 0}$ CTMC :

- $P(X_0)$
- $Q_X = (q_{i,j})_{i \leq n, j \leq n}$ intensity matrix

Properties

- $p_{i,j} = \frac{q_{i,j}}{-q_{i,i}}$ is the probability of transition from x_i to x_j
- $P(X_t) = P(X_O) \cdot exp(Q_X t)$ with $exp(M) = \sum_{n=0}^{\infty} \frac{M^n}{n!}$

Convergence

With some good conditions (ergodicity), $P(X_t) \underset{t \to \infty}{\longrightarrow} P(X^*)$

- Forward sampling :

$$\text{draw}(exp(-q_{i,i})) \text{ puis draw}((p_{i,j}, j \neq i)).$$

Properties

$(X_t \in x_1, \cdots, x_n)_{t \geq 0}$ CTMC :

- $P(X_0)$
- $Q_X = (q_{i,j})_{i \leq n, j \leq n}$ intensity matrix

Properties

- $p_{i,j} = \frac{q_{i,j}}{-q_{i,i}}$ is the probability of transition from x_i to x_j
- $P(X_t) = P(X_O) \cdot exp(Q_X t)$ with $exp(M) = \sum_{n=0}^{\infty} \frac{M^n}{n!}$

Convergence

With some good conditions (ergodicity), $P(X_t) \underset{t \to \infty}{\longrightarrow} P(X^*)$

- Forward sampling :

  draw($exp(-q_{i,i})$) puis draw($(p_{i,j}, j \neq i)$).
- Convergence of $P(X_t) = P(X_0) exp(Q_X t) \underset{t \to \infty}{\longrightarrow} P^*$

factorized CTMC : CTBN

CTBN

factorized CTMC : CTBN

CTBN

$(\mathbb{X}, G, (Q_X)_{X\in\mathbb{X}})$ Continuous-Time Bayesian Network if

factorized CTMC : CTBN

CTBN

$(\mathbb{X}, G, (Q_X)_{X \in \mathbb{X}})$ Continuous-Time Bayesian Network if

- $\mathbb{X} = (X_1, \cdots, X_n)$ continuous-time Markov process

factorized CTMC : CTBN

CTBN

$(\mathbb{X}, G, (Q_X)_{X\in\mathbb{X}})$ Continuous-Time Bayesian Network if

- $\mathbb{X} = (X_1, \cdots, X_n)$ continuous-time Markov process
- G oriented graph on $\mathbb{X}$

factorized CTMC : CTBN

CTBN

$(\mathbb{X}, G, (Q_X)_{X\in\mathbb{X}})$ Continuous-Time Bayesian Network if

- $\mathbb{X} = (X_1, \cdots, X_n)$ continuous-time Markov process
- G oriented graph on $\mathbb{X}$ (not DAG)

factorized CTMC : CTBN

CTBN

$(\mathbb{X}, G, (Q_X)_{X\in\mathbb{X}})$ Continuous-Time Bayesian Network if

- $\mathbb{X} = (X_1, \cdots, X_n)$ continuous-time Markov process
- G oriented graph on $\mathbb{X}$ (not DAG)
- $\forall X \in \mathbb{X}, Q_X$ conditional intensity matrix (CIM)

factorized CTMC : CTBN

CTBN

$(\mathbb{X}, G, (Q_X)_{X\in\mathbb{X}})$ Continuous-Time Bayesian Network if

- $\mathbb{X} = (X_1, \cdots, X_n)$ continuous-time Markov process
- G oriented graph on $\mathbb{X}$ (not DAG)
- $\forall X \in \mathbb{X}, Q_X$ conditional intensity matrix (CIM)

Q_X is a CIM $\iff \forall <pa_X>, Q_{X|<pa_X>}$ intensity matrix.

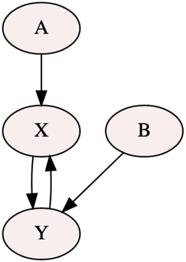
factorized CTMC : CTBN

CTBN

$(\mathbb{X}, G, (Q_X)_{X\in\mathbb{X}})$ Continuous-Time Bayesian Network if

- $\mathbb{X} = (X_1, \cdots, X_n)$ continuous-time Markov process
- G oriented graph on $\mathbb{X}$ (not DAG)
- $\forall X \in \mathbb{X}, Q_X$ conditional intensity matrix (CIM)

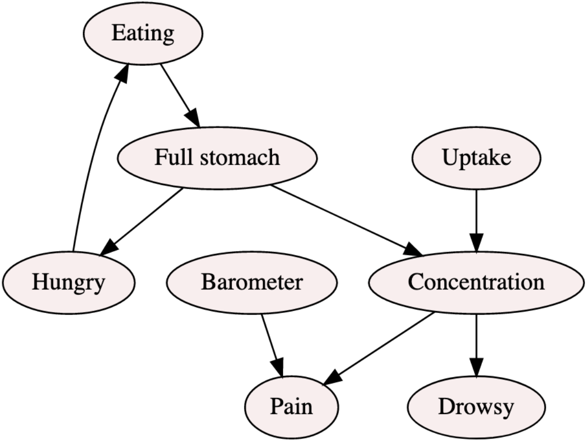
Q_X is a CIM $\iff \forall <pa_X>, Q_{X|<pa_X>}$ intensity matrix.



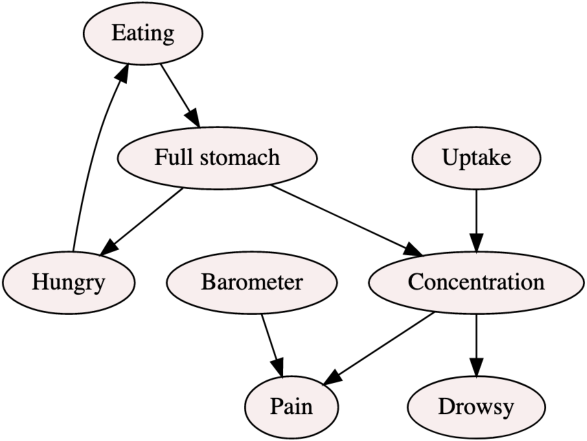
	A#j	
A#i	**a0**	**a1**
a0	-1.0000	1.0000
a1	2.0000	-2.0000

			X#j	
Y	**A**	**X#i**	**x0**	**x1**
y0	**a0**	**x0**	-1.0000	1.0000
		x1	2.0000	-2.0000
	a1	**x0**	-10.0000	10.0000
		x1	20.0000	-20.0000
y1	**a0**	**x0**	-5.0000	5.0000
		x1	6.0000	-6.0000
	a1	**x0**	-50.0000	50.0000
		x1	60.0000	-60.0000

CTBN - properties

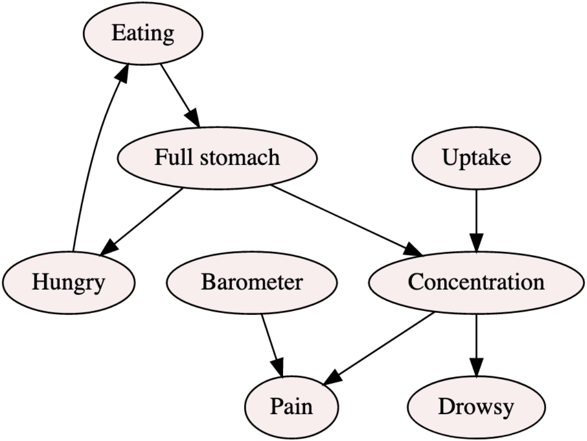


CTBN - properties



- Arc always temporal (which allows 'cycle') :
  $A \to B \iff P(B_{t+\delta_t} \mid A_t, \cdots) = \cdots$

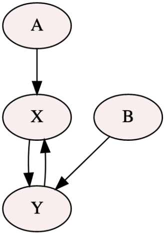
CTBN - properties

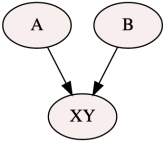


- Arc always temporal (which allows 'cycle') :
  $A \rightarrow B \iff P(B_{t+\delta_t} \mid A_t, \cdots) = \cdots$
- **A Continuous-Time Bayseian network is a joint continuous-time Markov process.**

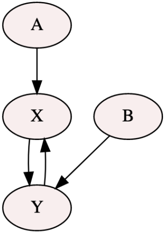
From a CTBN to the Markov process : amalgamation

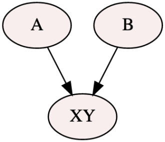
From a CTBN to the Markov process : amalgamation





From a CTBN to the Markov process : amalgamation

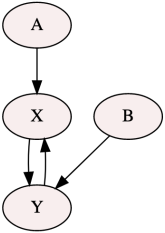


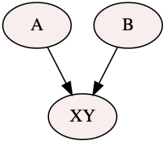


$$\mathbf{Q}_{X|a_0,y_0} = \begin{array}{c c} & \begin{array}{cc} x_0 & x_1 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \end{array} & \begin{pmatrix} -1 & 1 \\ 2 & -2 \end{pmatrix} \end{array} \quad \mathbf{Q}_{Y|b_1,x_0} = \begin{array}{c c} & \begin{array}{cc} y_0 & y_1 \end{array} \\ \begin{array}{c} y_0 \\ y_1 \end{array} & \begin{pmatrix} -3 & 3 \\ 4 & -4 \end{pmatrix} \end{array}$$

$$\mathbf{Q}_{X|a_0,y_1} = \begin{array}{c c} & \begin{array}{cc} x_0 & x_1 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \end{array} & \begin{pmatrix} -5 & 5 \\ 6 & -6 \end{pmatrix} \end{array} \quad \mathbf{Q}_{Y|b_1,x_1} = \begin{array}{c c} & \begin{array}{cc} y_0 & y_1 \end{array} \\ \begin{array}{c} y_0 \\ y_1 \end{array} & \begin{pmatrix} -7 & 7 \\ 8 & -8 \end{pmatrix} \end{array}$$

From a CTBN to the Markov process : amalgamation



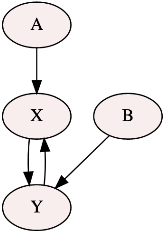


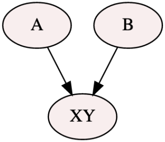
$$\mathbf{Q}_{X|a_0,y_0} = \begin{array}{c c} & \begin{array}{cc} x_0 & x_1 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \end{array} & \begin{pmatrix} -1 & 1 \\ 2 & -2 \end{pmatrix} \end{array} \quad \mathbf{Q}_{Y|b_1,x_0} = \begin{array}{c c} & \begin{array}{cc} y_0 & y_1 \end{array} \\ \begin{array}{c} y_0 \\ y_1 \end{array} & \begin{pmatrix} -3 & 3 \\ 4 & -4 \end{pmatrix} \end{array}$$

$$\mathbf{Q}_{X|a_0,y_1} = \begin{array}{c c} & \begin{array}{cc} x_0 & x_1 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \end{array} & \begin{pmatrix} -5 & 5 \\ 6 & -6 \end{pmatrix} \end{array} \quad \mathbf{Q}_{Y|b_1,x_1} = \begin{array}{c c} & \begin{array}{cc} y_0 & y_1 \end{array} \\ \begin{array}{c} y_0 \\ y_1 \end{array} & \begin{pmatrix} -7 & 7 \\ 8 & -8 \end{pmatrix} \end{array}$$

$$\mathbf{Q}_{XY|a_0,b_1} = \begin{array}{c c} & \begin{array}{cccc} (x_0,y_0) & (x_0,y_1) & (x_1,y_0) & (x_1,y_1) \end{array} \\ \begin{array}{c} (x_0,y_0) \\ (x_0,y_1) \\ (x_1,y_0) \\ (x_1,y_1) \end{array} & \begin{pmatrix} -4 & 3 & 1 & 0 \\ 4 & -9 & 0 & 5 \\ 2 & 0 & -9 & 7 \\ 0 & 6 & 8 & -14 \end{pmatrix} \end{array}$$

From a CTBN to the Markov process : amalgamation





$$\mathbf{Q}_{X|a_0,y_0} = \begin{array}{c c} & \begin{array}{cc} x_0 & x_1 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \end{array} & \begin{pmatrix} -1 & 1 \\ 2 & -2 \end{pmatrix} \end{array} \quad \mathbf{Q}_{Y|b_1,x_0} = \begin{array}{c c} & \begin{array}{cc} y_0 & y_1 \end{array} \\ \begin{array}{c} y_0 \\ y_1 \end{array} & \begin{pmatrix} -3 & 3 \\ 4 & -4 \end{pmatrix} \end{array}$$

$$\mathbf{Q}_{X|a_0,y_1} = \begin{array}{c c} & \begin{array}{cc} x_0 & x_1 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \end{array} & \begin{pmatrix} -5 & 5 \\ 6 & -6 \end{pmatrix} \end{array} \quad \mathbf{Q}_{Y|b_1,x_1} = \begin{array}{c c} & \begin{array}{cc} y_0 & y_1 \end{array} \\ \begin{array}{c} y_0 \\ y_1 \end{array} & \begin{pmatrix} -7 & 7 \\ 8 & -8 \end{pmatrix} \end{array}$$

$$\mathbf{Q}_{XY|a_0,b_1} = \begin{array}{c c} & \begin{array}{cccc} (x_0,y_0) & (x_0,y_1) & (x_1,y_0) & (x_1,y_1) \end{array} \\ \begin{array}{c} (x_0,y_0) \\ (x_0,y_1) \\ (x_1,y_0) \\ (x_1,y_1) \end{array} & \begin{pmatrix} -4 & 3 & 1 & 0 \\ 4 & -9 & 0 & 5 \\ 2 & 0 & -9 & 7 \\ 0 & 6 & 8 & -14 \end{pmatrix} \end{array}$$

The amalgamation of all the CIMs of a CTBN produces the intensity matrix of the joint Markov process.

Forward sampling dans un CTBN

Forward sampling dans un CTBN

CTBN Forward sampling

Forward sampling dans un CTBN

CTBN Forward sampling

1. $(X_{t+1}, D_t) = (argmin, min)_{X \in CTBN} Draw(Q_X)$

Forward sampling dans un CTBN

CTBN Forward sampling

1. $(X_{t+1}, D_t) = (argmin, min)_{X \in CTBN} Draw(Q_X)$
2. $x_{t+1} = Draw(Q_X)$

Forward sampling dans un CTBN

CTBN Forward sampling

1. $(X_{t+1}, D_t) = (argmin, min)_{X \in CTBN} Draw(Q_X)$
2. $x_{t+1} = Draw(Q_X)$
3. Note (X_t, D_t)

Forward sampling dans un CTBN

CTBN Forward sampling

1. $(X_{t+1}, D_t) = (argmin, min)_{X \in CTBN} Draw(Q_X)$
2. $x_{t+1} = Draw(Q_X)$
3. Note (X_t, D_t)
4. loop until stop

Forward sampling dans un CTBN

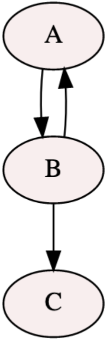
CTBN Forward sampling

1. $(X_{t+1}, D_t) = (argmin, min)_{X \in CTBN} Draw(Q_X)$
2. $x_{t+1} = Draw(Q_X)$
3. Note (X_t, D_t)
4. loop until stop

Forward sampling dans un CTBN

CTBN Forward sampling

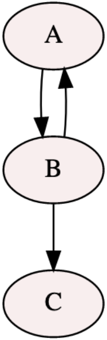
1. $(X_{t+1}, D_t) = (argmin, min)_{X \in CTBN} Draw(Q_X)$
2. $x_{t+1} = Draw(Q_X)$
3. Note (X_t, D_t)
4. loop until stop

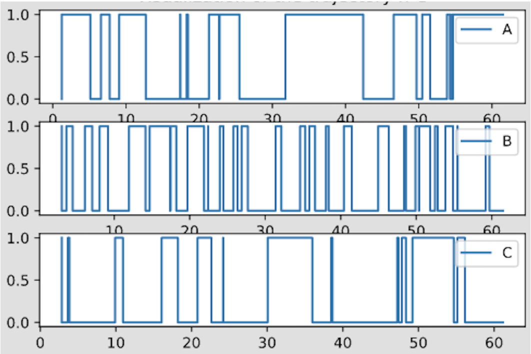


Forward sampling dans un CTBN

CTBN Forward sampling

1. $(X_{t+1}, D_t) = (argmin, min)_{X \in CTBN} Draw(Q_X)$
2. $x_{t+1} = Draw(Q_X)$
3. Note (X_t, D_t)
4. loop until stop





Implémentation (rapide)

Goal1 : way to define a CTBN

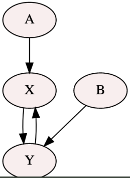
Goal1 : way to define a CTBN

```python
ctbn = Ctbn()
ctbn.add(gum.LabelizedVariable("A", "A", ["a0", "a1"]))
ctbn.add(gum.LabelizedVariable("B", "B", ["b0", "b1"]))
ctbn.add(gum.LabelizedVariable("X", "X", ["x0", "x1"]))
ctbn.add(gum.LabelizedVariable("Y", "Y", ["y0", "y1"]))

ctbn.addArc("A", "X")
ctbn.addArc("Y", "X")
ctbn.addArc("B", "Y")
ctbn.addArc("X", "Y")

ctbn.CIM("A")[:] = [[-1, 1],
                    [2, -2]]
ctbn.CIM("B")[:] = [[-3, 3],
                    [2, -2]]
ctbn.CIM("X")[{"A": "a0", "Y": "y0"}] = [[-1, 1],
                                         [2, -2]]
ctbn.CIM("X")[{"A": "a1", "Y": "y0"}] = [[-1, 1],
                                         [3, -3]]
ctbn.CIM("X")[{"A": "a0", "Y": "y1"}] = [[-5, 5],
                                         [6, -6]]
ctbn.CIM("X")[{"A": "a1", "Y": "y1"}] = [[-5, 5],
                                         [4, -4]]
ctbn.CIM("Y")[{"B": "b0", "X": "x0"}] = [[-2, 2],
                                         [5, -5]]
ctbn.CIM("Y")[{"B": "b1", "X": "x0"}] = [[-3, 3],
                                         [4, -4]]
ctbn.CIM("Y")[{"B": "b0", "X": "x1"}] = [[-3, 3],
                                         [5, -5]]
ctbn.CIM("Y")[{"B": "b1", "X": "x1"}] = [[-7, 7],
                                         [8, -8]]

gnb.sideBySide(ctbn,ctbn.CIM("A")._pot,ctbn.CIM("X")._pot)
```



	A#j	
A#i	a0	a1
a0	-1.0000	1.0000
a1	2.0000	-2.0000

			X#j	
Y	A	X#i	x0	x1
y0	a0	x0	-1.0000	1.0000
		x1	2.0000	-2.0000
	a1	x0	-1.0000	1.0000
		x1	3.0000	-3.0000
y1	a0	x0	-5.0000	5.0000
		x1	6.0000	-6.0000
	a1	x0	-5.0000	5.0000
		[illegible]	[illegible]	[illegible]

Goal2 : implement amalgamation

Goal2 : implement amalgamation

```python
im=CIM()
for x in ctbn.names():
    im*=ctbn.CIM(x)
im.to_matrix()
```

```
array([[ -7.,   1.,   3.,   0.,   1.,   0.,   0.,   0.,   2.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.],
       [  2.,  -8.,   0.,   3.,   0.,   1.,   0.,   0.,   0.,   2.,   0.,
          0.,   0.,   0.,   0.,   0.],
       [  2.,   0.,  -7.,   1.,   0.,   0.,   1.,   0.,   0.,   0.,   3.,
          0.,   0.,   0.,   0.,   0.],
       [  0.,   2.,   2.,  -8.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,
          3.,   0.,   0.,   0.,   0.],
       [  2.,   0.,   0.,   0.,  -9.,   1.,   3.,   0.,   0.,   0.,   0.,
          0.,   3.,   0.,   0.,   0.],
       [  0.,   3.,   0.,   0.,   2., -11.,   0.,   3.,   0.,   0.,   0.,
          0.,   0.,   3.,   0.,   0.],
       [  0.,   0.,   2.,   0.,   2.,   0., -12.,   1.,   0.,   0.,   0.,
          0.,   0.,   0.,   7.,   0.],
       [  0.,   0.,   0.,   3.,   0.,   2.,   2., -14.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   7.],
       [  5.,   0.,   0.,   0.,   0.,   0.,   0.,   0., -14.,   1.,   3.,
          0.,   5.,   0.,   0.,   0.],
       [  0.,   5.,   0.,   0.,   0.,   0.,   0.,   0.,   2., -15.,   0.,
          3.,   0.,   5.,   0.,   0.],
       [  0.,   0.,   4.,   0.,   0.,   0.,   0.,   0.,   2.,   0., -12.,
          1.,   0.,   0.,   5.,   0.],
       [  0.,   0.,   0.,   4.,   0.,   0.,   0.,   0.,   0.,   2.,   2.,
        -13.,   0.,   0.,   0.,   5.],
       [  0.,   0.,   0.,   0.,   5.,   0.,   0.,   0.,   6.,   0.,   0.,
          0., -15.,   1.,   3.,   0.],
       [  0.,   0.,   0.,   0.,   0.,   5.,   0.,   0.,   0.,   4.,   0.,
          0.,   2., -14.,   0.,   3.],
```

Goal3 : implement the 2 'inference'

Goal3 : implement the 2 'inference'

```
ie=SimpleCtbnInference(ctbn)
ie.makeInference()
gnb.sideBySide(ie.posterior("A"),ie.posterior("B"),ie.posterior("Y"),ie.posterior("X"))
```

A	
a0	a1
0.6667	0.3333

B	
b0	b1
0.4000	0.6000

Y	
y0	y1
0.6100	0.3900

X	
x0	x1
0.6065	0.3935

```
ie=ForwardSampleCtbnInference(ctbn)
ie.makeInference()
gnb.sideBySide(ie.posterior("A"),ie.posterior("B"),ie.posterior("Y"),ie.posterior("X"))
```

A	
a0	a1
0.6651	0.3349

B	
b0	b1
0.3968	0.6032

Y	
y0	y1
0.6081	0.3919

X	
x0	x1
0.6060	0.3940

Goal3 : implement the 2 'inference'

```
ie=SimpleCtbnInference(ctbn)
ie.makeInference()
gnb.sideBySide(ie.posterior("A"),ie.posterior("B"),ie.posterior("Y"),ie.posterior("X"))
```

A	
a0	a1
0.6667	0.3333

B	
b0	b1
0.4000	0.6000

Y	
y0	y1
0.6100	0.3900

X	
x0	x1
0.6065	0.3935

```
ie=ForwardSampleCtbnInference(ctbn)
ie.makeInference()
gnb.sideBySide(ie.posterior("A"),ie.posterior("B"),ie.posterior("Y"),ie.posterior("X"))
```

A	
a0	a1
0.6651	0.3349

B	
b0	b1
0.3968	0.6032

Y	
y0	y1
0.6081	0.3919

X	
x0	x1
0.6060	0.3940

Classes to code

Classes to code

- CIM

Classes to code

- CIM from `(gum.Potential)`

Classes to code

- CIM from `(gum.Potential)`
- CTBN,

Classes to code

- CIM from `(gum.Potential)`
- CTBN, `(gum.DiGraph,gum.DiscreteVariable,CIM)`

Classes to code

- CIM from `(gum.Potential)`
- CTBN, `(gum.DiGraph,gum.DiscreteVariable,CIM)`
- Convergence
  - 'Exact' method ($exp(IM)$)
  - Sampling (Forward Sampling)

CIM

CIM

CIM : wrapper de Potential

```python
def __init__(self, pot=None):
  if pot is None:
    self._pot = gum.Potential()
  else:
    self._pot = gum.Potential(pot)
  self._recordVars()

def add(self, v) -> "CIM":
  self._pot.add(v)
  self._recordVars()
  return self

def nbrDim(self) -> int:
  return self._pot.nbrDim()

def extract(self, ctxt) -> "CIM":
  return CIM(self._pot.extract(ctxt))

@property
def var_names(self):
  return self._pot.var_names

def __getitem__(self, i):
  return self._pot[i]

def __setitem__(self, i, v):
  self._pot[i] = v
```

CIM : Amalgamation

CIM : Amalgamation

Algorithm 2.2 Amalgamate two nodes of a CTBN.

Amalgamate(X, Y)

$\mathbf{Q}_{XY|\mathbf{Pa}(XY)} \leftarrow \emptyset$
for each $\langle pa_{Y \setminus X} \rangle \in \mathbf{pa}_{X \setminus Y}$ **and** $\langle pa_{X \setminus Y} \rangle \in \mathbf{pa}_{Y \setminus X}$
  $\mathbf{Q}^{XY} \leftarrow \mathbf{0}$
  for $i, j = 1, \ldots, |X|$ **and** $l, k = 1, \ldots, |Y|$
    $\mathbf{Q}^{X} \leftarrow \mathbf{Q}_{X|\langle pa_{X \setminus Y} \rangle, x_i}$
    $\mathbf{Q}^{Y} \leftarrow \mathbf{Q}_{Y|\langle pa_{Y \setminus X} \rangle, y_k}$
    if $i = j \wedge k = l$
      $q^{XY}_{(i,j),(k,l)} \leftarrow q^{X}_{i,j} + q^{Y}_{k,l}$
    else if $i = j \wedge k \neq l$
      $q^{XY}_{(i,j),(k,l)} \leftarrow q^{Y}_{k,l}$
    else if $i \neq j \wedge k = l$
      $q^{XY}_{(i,j),(k,l)} \leftarrow q^{X}_{i,j}$
    end if
  end for
  $\mathbf{Q}_{XY|\langle pa_{XY} \rangle} \leftarrow \mathbf{Q}^{XY}$
  $\mathbf{Q}_{XY|\mathbf{Pa}(XY)} \leftarrow \mathbf{Q}_{XY|\mathbf{Pa}(XY)} \cup \{\mathbf{Q}_{XY|\langle pa_{XY} \rangle}\}$
end for
return $\mathbf{Q}_{XY|\mathbf{Pa}(XY)}$

CIM : Amalgamation

Algorithm 2.2 Amalgamate two nodes of a CTBN.

$Amalgamate(X, Y)$

$\mathbf{Q}_{XY|\mathbf{Pa}(XY)} \leftarrow \emptyset$
for each $\langle pa_{Y\setminus X}\rangle \in \mathbf{pa}_{X\setminus Y}$ **and** $\langle pa_{X\setminus Y}\rangle \in \mathbf{pa}_{Y\setminus X}$
  $\mathbf{Q}^{XY} \leftarrow \mathbf{0}$
  for $i, j = 1, \ldots, |X|$ **and** $l, k = 1, \ldots, |Y|$
    $\mathbf{Q}^{X} \leftarrow \mathbf{Q}_{X|\langle pa_{X\setminus Y}\rangle, x_i}$
    $\mathbf{Q}^{Y} \leftarrow \mathbf{Q}_{Y|\langle pa_{Y\setminus X}\rangle, y_k}$
    if $i = j \wedge k = l$
      $q^{XY}_{(i,j),(k,l)} \leftarrow q^{X}_{i,j} + q^{Y}_{k,l}$
    else if $i = j \wedge k \neq l$
      $q^{XY}_{(i,j),(k,l)} \leftarrow q^{Y}_{k,l}$
    else if $i \neq j \wedge k = l$
      $q^{XY}_{(i,j),(k,l)} \leftarrow q^{X}_{i,j}$
    end if
  end for
  $\mathbf{Q}_{XY|\langle pa_{XY}\rangle} \leftarrow \mathbf{Q}^{XY}$
  $\mathbf{Q}_{XY|\mathbf{Pa}(XY)} \leftarrow \mathbf{Q}_{XY|\mathbf{Pa}(XY)} \cup \{\mathbf{Q}_{XY|\langle pa_{XY}\rangle}\}$
end for
return $\mathbf{Q}_{XY|\mathbf{Pa}(XY)}$

```python
    i = amal.instantiation()
    iX = cimX.instantiation()
    iY = cimY.instantiation()
    i.setFirst()
    while not i.end():
      iX.setVals(i)
      iY.setVals(i)
      for v in pXinY:
        iY.chgVal(v, iX[CIM.var_i(v.name())])
      for v in pYinX:
        iX.chgVal(v, iY[CIM.var_i(v.name())])
      dX = True
      for v in sX:
        if iX[CIM.var_i(v)] != iX[CIM.var_j(v)]:
          dX = False
          break
      dY = True
      for v in sY:
        if iY[CIM.var_i(v)] != iY[CIM.var_j(v)]:
          dY = False
          break
      if dX and dY:
        amal[i] = cimX[iX] + cimY[iY]
      elif dY:
        amal[i] = cimX[iX]
      elif dX:
        amal[i] = cimY[iY]
      else:
        amal[i] = 0

      i.inc()

    return amal
```

Implementations

- CIM : 210 lines
- CTBN
- Inférence
  - Simple
  - Sampling

Implementations

- CIM : 210 lines
- CTBN
- Inférence
  - Simple
  - Sampling

CTBN

Mainly : synchronization of 3 very differents objects :

CTBN

Mainly : synchronization of 3 very differents objects :

- oriented graph (`gum.DiGraph`),

CTBN

Mainly : synchronization of 3 very differents objects :

- oriented graph (`gum.DiGraph`),
- Discrete random variables in a dictionnary (`gum.DiscreteVariable`)

CTBN

Mainly : synchronization of 3 very differents objects :

- oriented graph (`gum.DiGraph`),
- Discrete random variables in a dictionnary (`gum.DiscreteVariable`)
- `CIMs` in a dictionnary.

CTBN

Mainly : synchronization of 3 very differents objects :

- oriented graph (`gum.DiGraph`),
- Discrete random variables in a dictionnary (`gum.DiscreteVariable`)
- `CIM`s in a dictionnary.

```python
def __init__(self):
    self.graph = gum.DiGraph()
    self.cim = {}
    self.id2var = {}
    self.name2id = {}

def add(self, var: gum.DiscreteVariable) -> NodeId:
    n = NodeId(self.graph.addNode())
    self.id2var[n] = var
    self.name2id[var.name()] = n

    self.bn0.add(var)

    v_i = var.clone()
    v_i.setName(CIM.var_i(var.name()))

    v_j = var.clone()
    v_j.setName(CIM.var_j(var.name()))

    self.cim[n] = CIM().add(v_j).add(v_i)

    return n

def addArc(self, val1: NameOrId, val2: NameOrId) -> Tuple[NodeId, Node
    n1 = self._nameOrId(val1)
    n2 = self._nameOrId(val2)

    self.graph.addArc(n1, n2)

    self.cim[n2].add(self.id2var[n1])

    return (n1, n2)
```

Implementations

- CIM : 210 lines
- CTBN : 190 lines
- Inférence
  - Simple
  - Sampling

Implementations

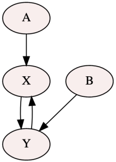
- CIM : 210 lines
- CTBN : 190 lines
- Inférence
  - Simple
  - Sampling

```
ctbn = Ctbn()
ctbn.add(gum.LabelizedVariable("A", "A", ["a0", "a1"]))
ctbn.add(gum.LabelizedVariable("B", "B", ["b0", "b1"]))
ctbn.add(gum.LabelizedVariable("X", "X", ["x0", "x1"]))
ctbn.add(gum.LabelizedVariable("Y", "Y", ["y0", "y1"]))

ctbn.addArc("A", "X")
ctbn.addArc("Y", "X")
ctbn.addArc("B", "Y")
ctbn.addArc("X", "Y")

ctbn.CIM("A")[:] = [[-1, 1],
                    [2, -2]]
ctbn.CIM("B")[:] = [[-3, 3],
                    [2, -2]]
ctbn.CIM("X")[{"A": "a0", "Y": "y0"}] = [[-1, 1],
                                         [2, -2]]
ctbn.CIM("X")[{"A": "a1", "Y": "y0"}] = [[-1, 1],
                                         [3, -3]]
ctbn.CIM("X")[{"A": "a0", "Y": "y1"}] = [[-5, 5],
                                         [6, -6]]
ctbn.CIM("X")[{"A": "a1", "Y": "y1"}] = [[-5, 5],
                                         [4, -4]]
ctbn.CIM("Y")[{"B": "b0", "X": "x0"}] = [[-2, 2],
                                         [5, -5]]
ctbn.CIM("Y")[{"B": "b1", "X": "x0"}] = [[-3, 3],
                                         [4, -4]]
ctbn.CIM("Y")[{"B": "b0", "X": "x1"}] = [[-3, 3],
                                         [5, -5]]
ctbn.CIM("Y")[{"B": "b1", "X": "x1"}] = [[-7, 7],
                                         [8, -8]]

gnb.sideBySide(ctbn,ctbn.CIM("A")._pot,ctbn.CIM("X")._pot)
```



	A#j	
A#i	a0	a1
a0	-1.0000	1.0000
a1	2.0000	-2.0000

			X#j	
Y	A	X#i	x0	x1
y0	a0	x0	-1.0000	1.0000
		x1	2.0000	-2.0000
	a1	x0	-1.0000	1.0000
		x1	3.0000	-3.0000
y1	a0	x0	-5.0000	5.0000
		x1	6.0000	-6.0000
	a1	x0	-5.0000	5.0000
		x1	4.0000	-4.0000

Inférence simple

Inférence simple

```python
class SimpleCtbnInference(CtbnInference):
    '''
    Exact inference using amalgamation to compute the Intensity Matrix corresponding to the ctbn
    (very bad for large models)
    '''

    def __init__(self, cim: Ctbn):
        super().__init__(cim)
        self._joint = None

    def makeInference(self):
        q = CIM()
        for nod in self._model.nodes():
            q = q.amalgamate(self._model.CIM(nod))

        q.from_matrix(expm(5000 * q.to_matrix()))

        t0 = gum.Potential()
        for n in q.var_names:
            if n[-1] == "i":
                t0.add(q._pot.variable(n))
        t0.fillWith(1).normalize()

        self._joint = (t0 * q._pot).margSumOut(t0.var_names)

    def posterior(self, name: str) -> gum.Potential:
        vj = CIM.var_j(name)
        return gum.Potential().add(self._model.variable(name)).fillWith(self._joint.margSumIn(vj), [vj])
```

Implémentations

- CIM : 210 lines
- CTBN : 190 lines
- Inférence
    - Simple : 23 lines
    - Sampling

Forward Sampling

Forward Sampling

Algorithm 2.1 Forward sample CTBN.

ForwardSample($\mathcal{N}$)

for each $X \in \mathcal{N}$
    choose $X(0)$ by sampling from $\mathcal{B}$
end for
$t \leftarrow 0$, $\sigma \leftarrow \emptyset$
repeat until termination
    $\mathit{Append}(\sigma, \langle X, t \rangle)$
    for each $X \in \mathcal{N}$
        if $\mathit{time}(X) \neq \mathit{null}$ **then** continue **end for**
        $\mathbf{A}_X \leftarrow \mathbf{A}_{X|\mathbf{Pa}(X)}$
        $i \leftarrow X(t)$
        $\Delta t \sim \text{Exponential}(a_{i,i})$
        $\mathit{time}(X) \leftarrow t + \Delta t$
    end for
    $X' \leftarrow \text{argmin}_{X \in \mathcal{N}} (\mathit{time}(X))$
    $t \leftarrow \mathit{time}(X')$
    $X(t) \sim \text{Multinomial}(\mathbf{A}_{X'}, X(t))$
    $\mathit{time}(X') = \mathit{null}$
    for each $Y \in \mathbf{Ch}(X')$
        $\mathit{time}(Y) = \mathit{null}$
    end for
end repeat
return σ

Forward Sampling

Algorithm 2.1 Forward sample CTBN.

ForwardSample($\mathcal{N}$)

for each $X \in \mathcal{N}$
    choose $X(0)$ by sampling from $\mathcal{B}$
end for
$t \leftarrow 0$, $\sigma \leftarrow \emptyset$
repeat until termination
    $\mathit{Append}(\sigma, \langle X, t \rangle)$
    for each $X \in \mathcal{N}$
        if $\mathit{time}(X) \neq \mathit{null}$ **then** continue **end for**
        $\mathbf{A}_X \leftarrow \mathbf{A}_{X|\mathbf{Pa}(X)}$
        $i \leftarrow X(t)$
        $\Delta t \sim \text{Exponential}(a_{i,i})$
        $\mathit{time}(X) \leftarrow t + \Delta t$
    end for
    $X' \leftarrow \text{argmin}_{X \in \mathcal{N}} (\mathit{time}(X))$
    $t \leftarrow \mathit{time}(X')$
    $X(t) \sim \text{Multinomial}(\mathbf{A}_{X'}, X(t))$
    $\mathit{time}(X') = \mathit{null}$
    for each $Y \in \mathbf{Ch}(X')$
        $\mathit{time}(Y) = \mathit{null}$
    end for
end repeat
return σ

Conclusion

- CIM : 210 lines
- CTBN : 190 lines
- Inférence
  - Simple : 23 lines
  - Sampling 90 lines

Conclusion

- CIM : 210 lines
- CTBN : 190 lines
- Inférence
  - Simple : 23 lines
  - Sampling 90 lines

Conclusion

- CIM : 210 lines
- CTBN : 190 lines
- Inférence
  - Simple : 23 lines
  - Sampling 90 lines

Code based on the result of a student project (L2).

Conclusion

- CIM : 210 lines
- CTBN : 190 lines
- Inférence
  - Simple : 23 lines
  - Sampling 90 lines

Code based on the result of a student project (L2).

goals reached !

Conclusion

- CIM : 210 lines
- CTBN : 190 lines
- Inférence
  - Simple : 23 lines
  - Sampling 90 lines

Code based on the result of a student project (L2).

goals reached !

- model CTBN : compact representation of continuous-time Markov processes with a very large state space .

Conclusion

- CIM : 210 lines
- CTBN : 190 lines
- Inférence
  - Simple : 23 lines
  - Sampling 90 lines

Code based on the result of a student project (L2).

goals reached !

- model CTBN : compact representation of continuous-time Markov processes with a very large state space .
- pyAgrum : toolbox for the implementation of new graphical models.

And now ?

- aGrUM/pyAgrum still a lab/academic tool. We will not stop maintaining & developing !
- Many users imply many responsabilities
  - Interaction
    gitlab issues, discord, gitter, linkedin, researchGate, what else ?
  - Structuration
    communauty (?), consortium (?)
  - Scientific orientation ?
    - models
    - algorithms
    - scientific committee
    - ?
  - Development orientation ?
    - weaknesses, strengths
    - missing features
    - Ragrum, JSagrum
    - Steering committee
    - ?