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 ⇒ No IDE).

Goals 1 to 3 make aGrUM interesting enough for outside the laboratory.

pyAgrum as a solution.

(< 6 years) 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 → local git → local gitlab → gitlab.

Open Source: GPL then LGPL.
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 ⇒ No IDE).
2. C++ !!!

Goals 1 to 3 rather well achieved but 4 was (at least) questionable.
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 → local git → local gitlab → gitlab

Open Source : GPL then LGPL.
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 ⇒ 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 ⇒ No IDE).
 2. C++ ! !!
 3. Optimized (as much as possible).
 4. Usable and improvable by others than us

goals 1 to 3 rather well achieved but 4 was (at least) questionable.

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 → local git → local gitlab → gitlab
4. Open Source : GPL then LGPL.
(> 10 years) aGrUM’s goals (as a tool for laboratory)

1. PGM as a library (and not as a software ⇒ 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 ⇒ 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
> 10 years) aGrUM’s goals (as a tool for laboratory)

1. PGM as a library (and not as a software ⇒ 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) questionable.
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 ⇒ 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) questionable.
- 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 ⇒ 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) questionable.
- 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 ⇒ 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) questionable.
• Goals 1 to 3 make aGrUM interesting enough for outside the laboratory.
• pyAgrum as a solution.

(< 6 years) 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.
4. Open Source: GPL then LGPL.
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) questionable.
- 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 ⇒ 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) questionable.
- 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 ⇒ 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) questionable.
- 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 ⇒ 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) questionable.
- 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 → local git → local gitlab → gitlab
- Open Source: GPL then LGPL.
aGrUM/pyAgrum: a framework for modeling, learning and using (probabilistic) graphical models

aGrUM (C++20)

pyAgrum (python(3))

pyAgrum.lib notebook
dynamicBN etc.

pyAgrum.skbn

pyAgrum.causal

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

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

aGrUM (C++20)

Bayesian networks

Model

Inference

pyAgrum (python(3))

pyAgrum.lib

notebook
dynamicBN

e等。

pyAgrum.skbn

pyAgrum.causal

Introduction, introspection, illustration
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 (C++20)

Core

Bayesian networks

- Model
- Inference
- Statistical learning

pyAgrum (python(3))

pyAgrum.lib

- notebook
- dynamicBN etc.

pyAgrum.causal

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

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

Core

- Random variables
- High-dimensional proba.

Bayesian networks

- Model
- Inference
- Statistical learning

Advanced Models

- FMDP
- Credal networks
- Influenced diagrams
- Markov networks
- PRM

Experimental

- pyAgrum.lib
- notebook
dynamicBN etc.

aGrUM

(C++20)
aGrUM/pyAgrum: a framework for modeling, learning and using (probabilistic) graphical models

- Core
  - Random variables
  - High-dimensional proba.
- Bayesian networks
  - Model
  - Inference
  - Statistical learning
- aGrUM (C++20)
- pyAgrum.lib
- pyAgrum.skbn
- pyAgrum.causal
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

- Core
- Random variables
- High-dimensional proba.
- Graphs

Bayesian networks
- Model
- Inference
- Statistical learning
- Credal networks

Advanced Models
- PRM

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

**aGrUM**

(C++20)

**Random variables**

**High-dimensional proba.**

**Graphs**

**Core**

**Bayesian networks**

**Model**

**Inference**

**Statistical learning**

**Advanced Models**

**Credal networks**

**Influence diagrams**

**Markov networks**

**PRM**

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

Core
- Random variables
- High-dimensional proba.
- Graphs

Bayesian networks
- Model
- Inference
- Statistical learning

Advanced Models
- FMDP
- Credal networks
- Influence diagrams
- Markov networks
- PRM

pyAgrum (C++20)
pyAgrum.lib
notebook
dynamicBN
etc.

pyAgrum.skbn
pyAgrum.causal

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

- **Core**
  - Random variables
  - High-dimensional proba.
  - Graphs

- **Bayesian networks**
  - Model
  - Inference
  - Statistical learning
  - FMDP
  - Credal networks
  - Influence diagrams
  - Markov networks
  - PRM

- **Advanced Models**
  - Experimental

- **pyAgrum (Python 3)**
  - pyAgrum.lib
  - Notebook
  - dynamicBN
  - etc.

- **pyAgrum (C++20)**
  - Core
  - Advanced Models

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

Core
- Random variables
- High-dimensional proba.
- Graphs

Bayesian networks
- Model
- Inference
- Statistical learning

Advanced Models
- FMDP
- Credal networks
- Influence diagrams
- Markov networks
- PRM

Experimental

pyAgrum
(python(3))

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

- **Core**
  - Random variables
  - High-dimensional proba.
  - Graphs

- **aGrUM (C++20)**
  - Advanced Models
    - FMDP
    - Credal networks
    - Influence diagrams
    - Markov networks
    - PRM

- **Bayesian networks**
  - Model
  - Inference
  - Statistical learning

- **pyAgrum (python(3))**
  - Experimental
  - Experimental
  - Dynamic BN
  - etc.

Introduction, introspection, illustration
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 (C++20)

Random variables

High-dimensional proba.

Graphs

Core

Bayesian networks

Model

Inference

Statistical learning

Advanced Models

FMDP

Credal networks

Influence diagrams

Markov networks

PRM

Experimental

pyAgrum.lib

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

- **Core**
  - Random variables
  - High-dimensional proba.
  - Graphs

- **Bayesian networks**
  - Model
  - Inference
  - Statistical learning

- **Advanced Models**
  - FMDP
  - Credal networks
  - Influence diagrams
  - Markov networks
  - PRM

- **Experimental**

- **pyAgrum**
  - (C++20)
  - pyAgrum.lib
  - notebook

Introduction, introspection, illustration
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
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 adhered by the LGPL v3, please contact us.
aGrUM is a C++ library designed for easily building applications using graphical models such as Bayesian networks, influence diagrams, decision trees, GAI networks or Markov decision processes.
Project description

Description: pyAgrum is a scientific C++ and Python library dedicated to Bayesian Networks and other Probabilistic Graphical Models. It provides a high-level interface to the port of the C++ libAgrum library allowing to create, model, learn, and 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 libAgrum API. The module is mainly generated by the libAgrum interface generator.
Project description

Description: pyAgrum is a scientific C++ and Python library dedicated to Bayesian Networks and other Probabilistic Graphical Models. It provides a high-level interface to the OpenBayes C++ library 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 OpenBayes API. The module is mainly generated by the SWIG/interflex generator.

conda-forge / packages / pyagrum 0.14.2

A wrapper for the Agrum library, to make flexible and scalable probabilistic graphical models.

Installers

conda install -c conda-forge pyagrum
conda install -c conda-forge label/gpc7 pyagrum
conda install -c conda-forge label/cf201991 pyagrum
pyAgrum is a Python wrapper for the C++ aGrUM library. 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. The module is mainly generated by the SWIG interface generator.

A wrapper for the Agrum library, to make flexible and scalable probabilistic graphical models.

To install this package with conda run one of the following:
conda install -c conda-forge pyagrum
conda install -c conda-forge/a agrum
conda install -c conda-forge label/a grum/p yagrum
conda install -c conda-forge label/cf2019001 pyagrum
conda install -c conda-forge label/cf2019003 pyagrum
Introduction, introspection, illustration
Introduction, introspection, illustration
Lazy Propagation

Lazy Propagation is the main exact inference for classical Bayesian networks in aGrUM/pyAgrum.

Class used for Lazy Propagation

LazyPropagation(bn) -> LazyPropagation

Parameters:

- bn (pyAgrum.BayesNet) - a Bayesian network

Returns

- pyAgrum.UndefinedElement - If no Bayes net has been assigned to the inference.

H(\text{arg})

Parameters

- X (int) - a node id
- modelName (str) - a node name

Returns

- the computed Shannon's entropy of a node gives the observation

Code quality in aGrUM/pyAgrum: documentation

Introduction, introspection, illustration
Lazy Propagation

Lazy Propagation is the main exact inference for classical Bayesian networks in aGrUM/pyAgrum.

Parameters:
- bn (pyAgrum.BayesNet) – a Bayesian network
- H
  Returns:
  A constant reference over the BayesNet referenced by this class.
- returnType
  Returns:
  The computed Shannon’s entropy of a node gives the observation.

Creating your first Bayesian network with pyAgrum

(Tutorial based on an OpenBayes [closed] website tutorial)

A Bayesian network (BN) is composed of random variables (nodes) and their conditional dependence (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 parent DAG.

```
piSprinkler = pyAgrum.CPT(pyAgrum.BayesNet()
```

```
P(Cloudy) = 0.1
P(Sprinkler | Cloudy) = 0.5
P(WetGrass | Sprinkler, Rain) = 0.8
```

```
P(Clear) = 0.9
P(Sprinkler | Clear) = 0.2
P(WetGrass | Clear) = 0.3
```

Lazy Propagation:

```
# Lazy Propagation
bn = pyAgrum.BayesNet()
bn.add_nodes(['Sprinkler', 'Cloudy', 'WetGrass', 'Rain'])
bn.add_edges(['Sprinkler', 'WetGrass', 'Sprinkler', 'Rain'])
bn.add_cpt('Sprinkler', values=[0, 1], ef=piSprinkler)
bn.add_cpt('Cloudy', values=[0, 1], ef=piCloudy)
bn.add_cpt('WetGrass', values=[0, 1], ef=piWetGrass)
bn.add_cpt('Rain', values=[0, 1], ef=piRain)
```

```
bn.inference('Sprinkler', 'WetGrass', 'Rain')
```

```
bn.compute('WetGrass', 'Rain', 'Sprinkler', 'Cloudy')
```

```
bn.inference('Sprinkler', 'WetGrass', 'Rain').show_cpt()
```

```
bn.inference('Sprinkler', 'WetGrass', 'Rain').show_edges()
```
Code quality in aGrUM/pyAgrum: tests

Introduction, introspection, illustration

# Profiling: 1402 ms

Failed 0 of 119 tests

Success rate: 100%

pyAgrum on Python 3.7.2 - linux

pyAgrum path: /home/phw/Workplace/docs/gitlab/agrum-dev/build/releases/wrappers/pyAgrum/__init__.py

Errors: 0

Time spent in tests: 2.786 s, make: 191.218 s and port: 1.832 s
**Code quality in aGrUM/pyAgrum: continuous integration**

### CI on different platforms

[Image of GitLab interface showing pipelines, build statuses, and nightly builds]

- **Nightly build (and tests)**
  - #50671123 by master → cfb6c2d2 → passed
  - macOS build → passed
  - Windows build → passed

### Deployment (to be continued)

- **Update message in testsOnPython.py**
  - 3 jobs from 8.14.2 in 18 minutes and 38 seconds (queued for 1 second)
  - Next

- **Pipeline #49619229**
  - Passed
  - Triggered 6 days ago by Pierre-Henri Willemin

---

*Introduction, introspection, illustration*
Some stats

Visits (readthedocs, agrum.org, notebooks)
Some stats

Visits (readthedocs, agrum.org, notebooks)
Some stats

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

![Map of Visits](image)

<table>
<thead>
<tr>
<th>Date Range</th>
<th>Unique Views</th>
<th>% of Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 févr. 2022 - 17 mars 2022</td>
<td><strong>5835</strong></td>
<td>100,00 %</td>
</tr>
<tr>
<td>16 févr. 2021 - 17 mars 2021</td>
<td><strong>3079</strong></td>
<td>100,00 %</td>
</tr>
</tbody>
</table>
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!
aGrUM/pyAgrum still a lab/academic tool. We will not stop maintaining & developing!

Many users imply many responsibilities
And now?

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

- Many users imply many responsibilities
  - Interaction
And now?

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

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

Many users imply many responsibilities

- 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

aGrUM (C++20)

Core
- Random variables
- High-dimensional proba.
- Graphs

Bayesian networks
- Model
- Inference
- Statistical learning

Advanced Models
- FMDP
- Credal networks
- Influence diagrams
- Markov networks
- PRM

pyAgrum (python(3))
- pyAgrum.lib
- notebook
- dynamicBN
- etc.
- pyAgrum.skbn
- pyAgrum.causal

Experimental
Introspection : focus on 3 elementary components

1. Core
2. Random variables
3. High-dimensional proba.
4. Graphs

pyAgrum (python(3))
Representation of Discrete Variable

[Content from the document]
**Representation of Discrete Variable**

**Data structure**: `DiscreteVariable`

- **goal**: map a finite domain \([0, \cdots, \text{domainSize}]\) 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:

- The kind of labels defines 4 different types of discrete variables:
  - LabelizedVariable: list of generic labels (as string),
  - RangeVariable: list of contiguous 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 $[0, \cdots, \text{domainSize}]$ 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:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>a</td>
<td>e</td>
<td>i</td>
<td>o</td>
<td>u</td>
<td>y</td>
</tr>
</tbody>
</table>

Introduction, introspection, illustration
**Data structure**: DiscreteVariable

**goal**: map a finite domain \([0, \cdots, \text{domainSize}]\) 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:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>a</td>
<td>e</td>
<td>i</td>
<td>o</td>
<td>u</td>
<td>y</td>
</tr>
</tbody>
</table>

The kind of labels defines 4 different types of discrete variables:
Representation of Discrete Variable

Data structure: DiscreteVariable

**goal**: map a finite domain \([0, \cdots, \text{domainSize}]\) 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:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>a</td>
<td>e</td>
<td>i</td>
<td>o</td>
<td>u</td>
<td>y</td>
</tr>
</tbody>
</table>

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 $[0, \cdots, \text{domainSize}]$ 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:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>a</td>
<td>e</td>
<td>i</td>
<td>o</td>
<td>u</td>
<td>y</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>a</td>
<td>e</td>
<td>i</td>
<td>o</td>
<td>u</td>
<td>y</td>
</tr>
</tbody>
</table>

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

- **LabelizedVariable** : list of generic labels (as string),
- **RangeVariable** : list of contiguous 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 $[0, \cdots, \text{domainSize}]$ 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:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>a</td>
<td>e</td>
<td>i</td>
<td>o</td>
<td>u</td>
<td>y</td>
</tr>
</tbody>
</table>

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

- **LabelizedVariable**: list of generic labels (as string),
- **RangeVariable**: list of contiguous 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:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>a</td>
<td>e</td>
<td>i</td>
<td>o</td>
<td>u</td>
<td>y</td>
</tr>
</tbody>
</table>

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

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

```latex
\begin{align*}
gum.LabelizedVariable & \quad A\{\text{Red}|\text{Green}|\text{Blue}\} \\
gum.RangeVariable & \quad A[3,6] \\
gum.DiscretizedVariable & \quad A[-1,-0.5,0.5,1,10] \\
gum.IntegerVariable & \quad A\{-14|5|6\}
\end{align*}
```
```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}

<table>
<thead>
<tr>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2159</td>
<td>0.1766</td>
<td>0.6075</td>
</tr>
</tbody>
</table>

---

gum.RangeVariable
A[3,6]

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>0.1729</td>
</tr>
</tbody>
</table>

---

gum.DiscretizedVariable
A[-1, -0.5, 0.5, 1, 10]

<table>
<thead>
<tr>
<th>[-1; -0.5]</th>
<th>[-0.5; 0.5]</th>
<th>[0.5; 1]</th>
<th>[1; 10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3127</td>
<td>0.3219</td>
<td>0.1325</td>
<td>0.2329</td>
</tr>
</tbody>
</table>

---

gum.IntegerVariable
A{-14|5|6}

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>-14</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>0.4087</td>
</tr>
</tbody>
</table>

---
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'
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 | Unoriented Graph

**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 | Unoriented Graph

**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.
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

```python
import pyAgrum

g = pyAgrum.DiGraph()
g.addNodes(5) & returns the generated nodeId

S = {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)
```

![Graph Diagram](attachment:diagram.png)
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
import pyAgrum as gum

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

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

![Digraph](image)

### UndiGraph

```python
import pyAgrum as gum

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
```

![UndiGraph](image)

### Mixed Graph

```python
import pyAgrum as gum

g = gum.MixedGraph()
g.addNodes(5)

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

![Mixed Graph](image)
Representation of multi-dimensional arrays

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

Implementing tensor algebra

Introduction, introspection, illustration
Representation of multi-dimensionnal arrays

**Data structure : Potential**

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

\[ f(,,) = g(,,) + h(,) \]

\[
\begin{array}{c}
[[[1.2.][3.4.]]][[5.6.][7.8.]] \\
g(,,)
\end{array}
\quad
\begin{array}{c}
[[1.3.][5.7.]] \\
h(,)
\end{array}
\quad
f(,,) = g(,,) + h(,)

\[ ? \]
Data structure: Potential

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

Implementing tensor algebra

\[
f(a, b, c) = g(b, a, c) + h(b, c)
\]

\[
\begin{array}{ccc}
[[[1. 2.] [3. 4.]] [5. 6.] [7. 8.]] & &
g(,,) \\
[[1. 3.] [5. 7.]] & & h(,) \\
\end{array}
\]

\[
f(,,) = g(,,) + h(,)
\]
**Data structure: Potential**

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

**Implementing tensor algebra**

\[ f(a, b, c) = g(b, a, c) + h(b, c) \]

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.0000</td>
<td>2.0000</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3.0000</td>
<td>4.0000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>5.0000</td>
<td>6.0000</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>7.0000</td>
<td>8.0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

\[ g(a, b, c) \]

\[ h(b, c) \]
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

```python
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

```python
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

```python
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

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

```
\begin{array}{c|c|c}
   & 0 & 1 \\
\hline
   A & 2.0000 & 5.0000 \\
   & 10.0000 & 13.0000 \\
   & 4.0000 & 7.0000 \\
   & 12.0000 & 15.0000 \\
\end{array}
```

```
\begin{array}{c|c|c}
   & 0 & 1 \\
\hline
   C & 7.0000 & 23.0000 \\
   & 11.0000 & 27.0000 \\
\end{array}
```

\[ \sum_{a,b,c} f(a,b,c) = 68.0 \]

\[ \sum_{b} f(a,b,c) \]
API for Potential

Quite complex API.

Creation and operations on Potential

```python
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

```

<table>
<thead>
<tr>
<th></th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2.0000</td>
</tr>
<tr>
<td>1</td>
<td>10.0000</td>
</tr>
</tbody>
</table>

\[ \sum_{a,b,c} f(a, b, c) \]
```

```

<table>
<thead>
<tr>
<th></th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>7.0000</td>
</tr>
<tr>
<td>1</td>
<td>11.0000</td>
</tr>
</tbody>
</table>

\[ \sum_{b} f(a, b, c) \]
```
Potential for probabilities

\[ P(A, B, C, D) = P(A) \times P(D) \times P(B|A, D) \times P(C|B) \]
Potential for probabilities

\[ P(A, B, C, D) = P(A) \times P(D) \times P(B | A, D) \times P(C | B) \]

Introduction, introspection, illustration
Potential for probabilities

$P(A, B, C, D) = P(A) \cdot P(D) \cdot P(B|A, D) \cdot P(C|B)$
Potential for probabilities

\[ P(A, B, C, D) = P(A) \times P(D) \times P(B|A, D) \times P(C|B) \]
Potential for probabilities

\[ P(A, B, C, D) = P(A) \times P(D) \times P(B|A, D) \times P(C|B) \]
Potential for probabilities (2)

\[
P(D|C) = P(D, C) P(C) = \sum_{A, B} P(A, B, C, D) \sum_{A, B, D} P(A, B, C, D)
\]
$P(D|C)$
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")
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0.7409</td>
<td>0.6943</td>
</tr>
<tr>
<td>1</td>
<td>0.2591</td>
<td>0.3057</td>
</tr>
</tbody>
</table>
Potential for probabilities (2)

\[ P(D | C) = \frac{P(D, C)}{P(C)} = \frac{\sum_{A,B} P(A, B, C, D)}{\sum_{A,B,D} P(A, B, C, D)} \]
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
Using these components to build new model in pyAgrum

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

dynamic Bayesian Network

Causal model
A dynamic Bayesian network is a Bayesian network with variables indexed by time $t$ and by $i$:

$$X(t) = X(t-1), \ldots, X(N)$$

for which those properties hold:

- **Markov property:**
  $$P(X(t) | X(0), \ldots, X(t-1)) = P(X(t) | X(t-1))$$

- **Homogeneity:**
  $$P(X(t) | X(0), \ldots, X(t-1)) = \cdots = P(X(1) | X(0))$$
A dynamic Bayesian network is a Bayesian network with variables indexed by the time $t$ and by $i$: $X^{(t)} = X_1^{(t)}, \ldots, X_N^{(t)}$. 

Markov property:

$P(X^{(t)} | X^{(0)}, \ldots, X^{(t-1)}) = P(X^{(t)} | X^{(t-1)})$.

Homogeneity:

$P(X^{(1)} | X^{(0)}) = \cdots = P(X^{(t)} | X^{(0)})$. 

---

dynamic Bayesian Networks
dBN (dynamic BN)

A dynamic Bayesian network is a Bayesian network with variables indexed by the time \( t \) and by \( i \): \( X^{(t)} = X^{(t)}_1, \ldots, X^{(t)}_N \) for which those properties hold:
a dynamic Bayesian network is a Bayesian network with variables indexed by the time $t$ and by $i$ : $X^{(t)} = X_1^{(t)}, \ldots, X_N^{(t)}$ for which those properties hold :

- Markov property :
a dynamic Bayesian network is a Bayesian network with variables indexed by the time $t$ and by $i$ : $X^{(t)} = X_1^{(t)}, \ldots, X_N^{(t)}$ for which those properties hold:

- **Markov property**:
  $$P(X^{(t)} \mid X^{(0)}, \ldots, X^{(t-1)}) = P(X^{(t)} \mid X^{(t-1)}),$$

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

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

- **Homogeneity**:
  $$P(X^{(t)} \mid X^{(t-1)}) = \cdots = P(X^{(1)} \mid X^{(0)}).$$
A dynamic Bayesian network is a Bayesian network with variables indexed by the time $t$ and by $i$ : $X^{(t)} = X_1^{(t)}, \ldots, X_N^{(t)}$ for which those properties hold:

- **Markov property**:
  \[
  P(X^{(t)} | X^{(0)}, \ldots, X^{(t-1)}) = P(X^{(t)} | X^{(t-1)}),
  \]

- **Homogeneity**:
  \[
  P(X^{(t)} | X^{(t-1)}) = \cdots = P(X^{(1)} | X^{(0)}).
  \]
A dynamic Bayesian network is defined by

- initial distributions \( P(X(0)) \),
- the transition between the variables at time \( t-1 \) and the same variables at time \( t \) (timeslices).

The representation a.k.a 2TBN (2 timeslices BN) allows the modeling of a virtually infinite Bayesian network which is the unrolled model from time 0.
A dynamic Bayesian network is defined by
- initial distributions \( P(X^{(0)}) \),
A dynamic Bayesian network is defined by

- initial distributions \( P(X^{(0)}) \),
- the transition between the variables at time \( t - 1 \) and the same variables at time \( t \) (timeslices).
A dynamic Bayesian network is defined by

- initial distributions \( P(X^{(0)}) \),
- the transition between the variables at time \( t - 1 \) and the same variables at time \( t \) (timeslices).
A dynamic Bayesian network is defined by

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

\[
P(x_1^{(t)}, \ldots, x_5^{(t)} \mid x_1^{(t-1)}, \ldots, x_5^{(t-1)})
\]
A dynamic Bayesian network is defined by

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

\[
P(x_1^{(t)}, \ldots, x_5^{(t)} \mid x_1^{(t-1)}, \ldots, x_5^{(t-1)}) = P(x_1^{(t)} \mid x^{(t-1)})
\]
Dynamic Bayesian networks: 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 variables at time \( t \) (timeslices).

\[
P(x_1^{(t)}, \ldots, x_5^{(t)} \mid x_1^{(t-1)}, \ldots, x_5^{(t-1)})
= P(x_1^{(t)} \mid x^{(t-1)})
\]

\[
P(x_2^{(t)} \mid x_1^{(t-1)}, x_2^{(t-1)}, x_3^{(t-1)})
\]
A dynamic Bayesian network is defined by

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

\[
P(x_1^{(t)}, \ldots, x_5^{(t)} \mid x_1^{(t-1)}, \ldots, x_5^{(t-1)})
= P(x_1^{(t)} \mid x^{(t-1)})
\]

\[
P(x_2^{(t)} \mid x_1^{(t-1)}, x_2^{(t-1)}, x_3^{(t-1)})
\]

\[
P(x_3^{(t)} \mid x_2^{(t-1)}, x_3^{(t-1)}, x_4^{(t-1)})
\]

\[
P(x_4^{(t)} \mid x_4^{(t-1)}, x_5^{(t-1)})
\]

1024 versus \( 4 + 16 + 16 + 8 + 2 = 46 \) !!
A dynamic Bayesian network is defined by

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

\[
P(x_1^{(t)}, \ldots, x_5^{(t)} | x_1^{(t-1)}, \ldots, x_5^{(t-1)})
\]

\[
= P(x_1^{(t)} | x^{(t-1)})
\]

\[
P(x_2^{(t)} | x_1^{(t-1)}, x_2^{(t-1)}, x_3^{(t-1)})
\]

\[
P(x_3^{(t)} | x_2^{(t-1)}, x_3^{(t-1)}, x_4^{(t-1)})
\]

\[
P(x_4^{(t)} | x_4^{(t-1)}, x_5^{(t-1)})
\]

\[
P(x_5^{(t)})
\]

1024 versus \( 4+16+16+8+2=46 \) !!
A dynamic Bayesian network is defined by

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

\[
P(x_1^{(t)}, \ldots, x_5^{(t)} | x_1^{(t-1)}, \ldots, x_5^{(t-1)})
= P(x_1^{(t)} | x^{(t-1)})
\]

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

a discrete variable \( (X_n) \) (at time \( n \)).

Parameters for this model:

- Initial condition: \( P(X_0) \)
- Transition probabilities: \( P(X_n | X_{n-1}) \)

Equivalent dynamic Bayesian network:

\[
P(X_n | X_{n-1}) = \begin{bmatrix}
0.25 & 0 & 0 \\
0.75 & 0 & 0 \\
0.25 & 0 & 0.5 \\
0.25 & 0 & 0.5 \\
0.25 & 0 & 0.25
\end{bmatrix}
\]

Could we do the same with continuous time?
Markov Chain and dynamic Bayesian network

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

Markov chain

- a discrete variable \( (X^n) \) (at time \( n \)).
- Parameters for this model:
  - Initial condition: \( P(X^0) \)
  - Transition probabilities: \( P(X^n | X^{n-1}) \)

Equivalent dynamic Bayesian network:
Markov Chain and dynamic Bayesian network

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

**Markov chain**
- a discrete variable \((X^n)\) (at time \(n\)).
- Parameters for this model:
  - Initial condition: \(P(X^0)\)
  - Transition probabilities: \(P(X^n | X^{n-1})\)

**Equivalent dynamic Bayesian network:**
- dBN: \(x^0 \rightarrow x^1 \rightarrow x^2 \rightarrow x^3 \rightarrow \ldots\)
- 2TBN: \(x^0 \rightarrow x^n\)

Introduction, introspection, illustration
Markov Chain and dynamic Bayesian network

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

Markov chain

- a discrete variable \((X^n)\) (at time \(n\)).
- Parameters for this model:
  - Initial condition: \(P(X^0)\)
  - Transition probabilities: \(P(X^n | X^{n-1})\)

Equivalent dynamic Bayesian network:

dBN: \(x^0 \rightarrow x^1 \rightarrow x^2 \rightarrow x^3 \rightarrow \ldots\)

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 \textit{any time},

\[
\begin{align*}
0 & \quad t_1 & \quad t_2 & \cdots \\
X(0) & \quad X(t_1) & \quad X(t_2)
\end{align*}
\]
Continuous-Time Markov Process

Dynamic processus dynamique verifying:

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

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

Continuous time Markov property:

\[ 0 \quad X(0) \quad t_1 \quad X(t_1) \quad t_2 \quad X(t_2) \quad t_3 \ldots \]
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

\[ D \sim \text{Exp}(\lambda) \]
- Cumulative distribution function
  \[ \forall d > 0, F(d) = 1 - e^{-\lambda d} \]
- \[ \mathbb{E}(D) = \lambda^{-1} \]
- \[ \sigma(D) = \lambda^{-1} \]
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:

\[ X \sim \text{Exp}(\lambda), Y \sim \text{Exp}(\mu), X | Y = \min(X, Y) \sim \text{Exp}(\lambda + \mu) \]

Continuous Time Markov Chain (CTMC) is defined by:

\[ P(X_0) \forall i, j \geq n, q_{i,j} \text{ such that } \]

1. \[ q_{i,i} \in \mathbb{R}^{-} \]
2. \[ q_{i,j} \in \mathbb{R}^{+} \]
3. \[ \sum_j q_{i,j} = 0 \]

\[ Q_X = \begin{bmatrix} -0.21 & 0 & 0 \\ 0 & -0.20 & 0 \\ 0 & 0 & -0.21 \end{bmatrix} \]

\[ q_{i,j} \text{ is the parameter of the exponential distribution controlling the transition from state } i \text{ to state } j \]

\[ q_{i,i} \text{ is the parameter of the exponential distribution controlling a transition from state } i. \]
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

\[ X \sim \text{Exp}(\lambda), \ Y \sim \text{Exp}(\mu), \ X \perp Y \Rightarrow \min(X, Y) \sim \text{Exp}(\lambda + \mu) \]
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

\[ X \sim \text{Exp}(\lambda), \ Y \sim \text{Exp}(\mu), \ X \perp \!\!\!\!\!\!\perp Y \Rightarrow \min(X, Y) \sim \text{Exp}(\lambda + \mu) \]

Continuous Time Markov Chain

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

\[ X \sim \text{Exp}(\lambda), \ Y \sim \text{Exp}(\mu), \ X \perp Y \Rightarrow \min(X, Y) \sim \text{Exp}(\lambda + \mu) \]

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 parameter 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

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

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

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

\[ P(X_t) = P(X_0) \cdot \exp(Q_X t) \text{ with } \exp(M) = \sum_{n=0}^{\infty} \frac{M^n}{n!} \]
\[(X_t \in x_1, \cdots, x_n)_{t \geq 0} \text{ CTMC:}\]

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

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

\[
P(X_t) = P(X_0) \cdot \exp(Q_X t) \text{ with } \exp(M) = \sum_{n=0}^{\infty} \frac{M^n}{n!}\]

**Convergence**

With some good conditions (ergodicity), \(P(X_t) \xrightarrow{t \to \infty} P(X^*)\)
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 \(x_j\)
- \(P(X_t) = P(X_0) \cdot \exp(Q_X t)\) with \(\exp(M) = \sum_{n=0}^{\infty} \frac{M^n}{n!}\)

Convergence

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

- Forward sampling:
  
  \begin{align*}
  &\text{draw}(\exp(-q_{i,i})) \quad \text{puis} \quad \text{draw}((p_{i,j}, j \neq i)).
  \end{align*}
Properties

\((X_t \in x_1, \ldots, x_n)_{t \geq 0}\) 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 \(x_j\)
- \(P(X_t) = P(X_0) \cdot \exp(Q_X t)\) with \(\exp(M) = \sum_{n=0}^{\infty} \frac{M^n}{n!}\)

Convergence

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

- Forward sampling:
  \[\text{draw}(\exp(-q_{i,i})) \text{ puis draw}((p_{i,j}, j \neq i))\].
- Convergence of \(P(X_t) = P(X_0)\exp(Q_X t) \xrightarrow{t \to \infty} P^*\)
factorized CTMC : CTBN

CTBN

Continuous-Time Bayesian Network if $X = (X_1, \ldots, X_n)$

- $G$ oriented graph on $X$ (not DAG)
- $Q_X$ conditional intensity matrix (CIM)

$Q_X$ is a CIM $\iff \forall \langle \text{pa}_X, Q_X | \text{pa}_X \rangle$ intensity matrix.
factorized CTMC : CTBN

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

$\forall X \in X$, $Q_X$ conditional intensity matrix (CIM) $\iff \forall <\text{pa}_X>, Q_X|_{<\text{pa}_X>}$ intensity matrix.
factorized CTMC : CTBN

CTBN

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

- $X = (X_1, \cdots, X_n)$ continuous-time Markov process
factorized CTMC : CTBN

\[ (\mathbf{X}, G, (Q_X)_{X \in \mathbf{X}}) \] Continuous-Time Bayesian Network if
- \( \mathbf{X} = (X_1, \ldots, X_n) \) continuous-time Markov process
- \( G \) oriented graph on \( \mathbf{X} \)
factorized CTMC : CTBN

CTBN

\((\mathbf{X}, G, (Q_X)_{X \in \mathbf{X}})\) Continuous-Time Bayesian Network if
- \(\mathbf{X} = (X_1, \cdots, X_n)\) continuous-time Markov process
- \(G\) oriented graph on \(\mathbf{X}\) (not DAG)
**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

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

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

\(Q_X\) is a CIM \(\iff\) \(\forall <pax>, Q_X|_{<pax>}\) intensity matrix.
factorized CTMC : CTBN

CTBN

\((X, G, (Q_X)_{X \in X})\) Continuous-Time Bayesian Network if
- \(X = (X_1, \cdots, X_n)\) continuous-time Markov process
- \(G\) oriented graph on \(X\) (not DAG)
- \(\forall X \in X, Q_X\) conditional intensity matrix (CIM)

\(Q_X\) is a CIM \(\iff\) \(\forall \langle pa_X \rangle, Q_{X|\langle pa_X \rangle}\) intensity matrix.

<table>
<thead>
<tr>
<th>(X#j)</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>A</td>
<td>X#i</td>
<td>x0</td>
<td>x1</td>
<td></td>
</tr>
<tr>
<td>y0</td>
<td>a0</td>
<td>x0</td>
<td>-1.0000</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>x1</td>
<td>2.0000</td>
<td>-2.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a1</td>
<td>x0</td>
<td>-10.0000</td>
<td>10.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>x1</td>
<td>20.0000</td>
<td>-20.0000</td>
<td></td>
</tr>
<tr>
<td>y1</td>
<td>a0</td>
<td>x0</td>
<td>-5.0000</td>
<td>5.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>x1</td>
<td>6.0000</td>
<td>-6.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a1</td>
<td>x0</td>
<td>-50.0000</td>
<td>50.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>x1</td>
<td>60.0000</td>
<td>-60.0000</td>
<td></td>
</tr>
</tbody>
</table>
A Continuous-Time Bayesien network is a joint continuous-time Markov process.
Arc always temporal (which allows 'cycle'):
\[ A \rightarrow B \iff P(B_{t+\delta_t} \mid A_t, \cdots) = \cdots \]
CTBN - properties

- Arc always temporal (which allows 'cycle'):
  \[ A \rightarrow B \iff P(B_{t+\delta_t} | A_t, \cdots) = \cdots \]

- A Continuous-Time Bayesian network is a joint continuous-time Markov process.
From a CTBN to the Markov process: amalgamation
From a CTBN to the Markov process: amalgamation

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

\[
Q_{X|a_0,y_0} = \begin{pmatrix}
    x_0 & x_1 \\
    2 & -2
\end{pmatrix}
\]

\[
Q_{X|a_0,y_1} = \begin{pmatrix}
    x_0 & x_1 \\
    6 & -6
\end{pmatrix}
\]

\[
Q_{Y|b_1,x_0} = \begin{pmatrix}
    y_0 & y_1 \\
    -3 & 3
\end{pmatrix}
\]

\[
Q_{Y|b_1,x_1} = \begin{pmatrix}
    y_0 & y_1 \\
    -7 & 7
\end{pmatrix}
\]
From a CTBN to the Markov process: amalgamation

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

Introduction, introspection, illustration

\[
\begin{align*}
Q_{X|a_0, y_0} &= x_0 \begin{pmatrix} -1 & 1 \\ 2 & -2 \end{pmatrix} & Q_{Y|b_1, x_0} &= y_0 \begin{pmatrix} -3 & 3 \\ 4 & -4 \end{pmatrix} \\
Q_{X|a_0, y_1} &= x_0 \begin{pmatrix} -5 & 5 \\ 6 & -6 \end{pmatrix} & Q_{Y|b_1, x_1} &= y_0 \begin{pmatrix} -7 & 7 \\ 8 & -8 \end{pmatrix}
\end{align*}
\]

\[
Q_{XY|a_0, b_1} = \begin{pmatrix}
(x_0, y_0) & (x_0, y_1) & (x_1, y_0) & (x_1, y_1) \\
(x_0, y_0) & -4 & 3 & 1 & 0 \\
(x_0, y_1) & 4 & -9 & 0 & 5 \\
(x_1, y_0) & 2 & 0 & -9 & 7 \\
(x_1, y_1) & 0 & 6 & 8 & -14
\end{pmatrix}
\]
The amalgamation of all the CIMs of a CTBN produces the intensity matrix of the joint Markov process.
Forward sampling dans un CTBN

\[ (X_{t+1}, D_t) = \arg \min_{X \in \text{CTBN}} \text{Draw}(Q_{X_{t+1}}) \]

Note \((X_t, D_t)\)

loop until stop
Forward sampling dans un CTBN

CTBN Forward sampling

1 \( (X_{t+1}, D_t) = \underset{X \in \text{CTBN}}{\text{argmin}} \ \text{min} \ \text{Draw}(Q_X) \)

2 \( x_{t+1} = \text{Draw}(Q_{X_{t+1}}) \)

3 Note \( (X_t, D_t) \)

4 loop until stop

Introduction, introspection, illustration
CTBN Forward sampling

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

1. \((X_{t+1}, D_t) = (\text{argmin}, \text{min})_{X \in \text{CTBN}} \text{Draw}(Q_x)\)
2. \(x_{t+1} = \text{Draw}(Q_x)\)
CTBN Forward sampling

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

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

1. \((X_{t+1}, D_t) = (\text{arg min}, \text{min})_{X \in \text{CTBN}} \text{Draw}(Q_X)\)
2. \(x_{t+1} = \text{Draw}(Q_X)\)
3. Note \((X_t, D_t)\)
4. loop until stop
Forward sampling dans un CTBN

CTBN Forward sampling

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

1. $(X_{t+1}, D_t) = (\arg\min, \min)_{X \in \text{CTBN}} \text{Draw}(Q_X)$
2. $x_{t+1} = \text{Draw}(Q_X)$
3. Note $(X_t, D_t)$
4. loop until stop
Implémentation (rapide)
Goal 1: way to define a CTBN
Goal 1: way to define a CTBN

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

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

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

gnb.sideBySide(c, c.CIM("A")_.pot, c.CIM("X")_.pot)
Goal2: implement amalgamation
Goal 2: 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.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  2.,  -8.,   0.,   0.,   1.,   0.,   0.,   0.,   2.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  2.,   0.,  -7.,   1.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   3.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   2.,   2.,  -8.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,
         3.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  2.,   0.,   0.,   0.,  -9.,   1.,   3.,   0.,   0.,   0.,   0.,   0.,
         0.,   3.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   3.,   0.,   0.,   2.,  -11.,   0.,   3.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   0.,   2.,   0.,   2.,  -12.,   1.,   0.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   7.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   3.,   0.,   2.,   2.,  -14.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   7.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  5.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,  -14.,   1.,   3.,
         0.,   5.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   5.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,  -15.,   0.,
         3.,   0.,   5.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   0.,   4.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   2.,   0.,
         1.,   0.,   0.,   5.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   4.,   0.,   0.,   0.,   0.,   0.,   0.,   2.,
         2.,   2.,  -13.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   5.,   0.,   0.,   0.,   6.,   0.,
         0.,   0.,   0.,   0.,   0.,   3.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   5.,   0.,   0.,   0.,   0.,   4.,   0.,
         0.,   2.,   -14.,   0.,   0.,   0.,   3.,   0.,   0.,   0.,   0.,   0.],
```
Goal 3: 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"))

<table>
<thead>
<tr>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>a1</td>
</tr>
<tr>
<td>0.6667</td>
<td>0.3333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b0</td>
<td>b1</td>
</tr>
<tr>
<td>0.4000</td>
<td>0.6000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>y0</td>
<td>y1</td>
</tr>
<tr>
<td>0.6100</td>
<td>0.3900</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>x1</td>
</tr>
<tr>
<td>0.6065</td>
<td>0.3935</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>a1</td>
</tr>
<tr>
<td>0.6651</td>
<td>0.3349</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b0</td>
<td>b1</td>
</tr>
<tr>
<td>0.3968</td>
<td>0.6032</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>y0</td>
<td>y1</td>
</tr>
<tr>
<td>0.6081</td>
<td>0.3919</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>x1</td>
</tr>
<tr>
<td>0.6060</td>
<td>0.3940</td>
</tr>
</tbody>
</table>
Goal 3: implement the 2 'inference'

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

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Y</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>0.6667</td>
<td>0.4000</td>
<td>0.6100</td>
<td>0.6065</td>
</tr>
<tr>
<td>a1</td>
<td>0.3333</td>
<td>0.6000</td>
<td>0.3900</td>
<td>0.3935</td>
</tr>
</tbody>
</table>

```

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

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Y</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>0.6651</td>
<td>0.3968</td>
<td>0.6081</td>
<td>0.6060</td>
</tr>
<tr>
<td>a1</td>
<td>0.3349</td>
<td>0.6032</td>
<td>0.3919</td>
<td>0.3940</td>
</tr>
</tbody>
</table>
```
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)
Introduction, introspection, illustration
```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 : wrapper de Potential**
Algorithm 2.2 Amalgamate two nodes of a CTBN.

\textbf{Amalgamate}(X, Y)

1: \( Q_{XY|\text{pa}(XY)} \leftarrow \emptyset \)
2: \textbf{for} each \( \langle \text{pa}_Y \setminus X \rangle \in \text{pa}_X \setminus Y \) and \( \langle \text{pa}_X \setminus Y \rangle \in \text{pa}_Y \setminus X \)
3: \( Q^{XY} \leftarrow 0 \)
4: \textbf{for} \( i, j = 1, \ldots, |X| \) and \( l, k = 1, \ldots, |Y| \)
5: \( Q^X \leftarrow Q^X|\langle \text{pa}_X \setminus Y \rangle, x_i \)
6: \( Q^Y \leftarrow Q^Y|\langle \text{pa}_Y \setminus X \rangle, y_k \)
7: \textbf{if} \( i = j \land k = l \)
8: \( q_{(i,j),(k,l)}^{XY} \leftarrow q_{i,j}^X + q_{k,l}^Y \)
9: \textbf{else if} \( i = j \land k \neq l \)
10: \( q_{(i,j),(k,l)}^{XY} \leftarrow q_{k,l}^Y \)
11: \textbf{else if} \( i \neq j \land k = l \)
12: \( q_{(i,j),(k,l)}^{XY} \leftarrow q_{i,j}^X \)
13: \textbf{end if}
14: \textbf{end for}
15: \( Q_{XY|\langle \text{pa}_X \setminus Y \rangle} \leftarrow Q^{XY} \)
16: \( Q_{XY|\text{pa}(XY)} \leftarrow Q_{XY|\text{pa}(XY)} \cup \{ Q_{XY|\langle \text{pa}_X \setminus Y \rangle} \} \)
17: \textbf{end for}
18: \textbf{return} \( Q_{XY|\text{pa}(XY)} \)
Algorithm 2.2 Amalgamate two nodes of a CTBN.

Amalgamate($X, Y$)

1: $Q_{XY|Pa(XY)} \leftarrow \emptyset$
2: for each $\langle pa_{Y\setminus X} \rangle \in pa_{X\setminus Y}$ and $\langle pa_{X\setminus Y} \rangle \in pa_{Y\setminus X}$
3:     $Q^{XY} \leftarrow 0$
4: for $i, j = 1, \ldots, |X|$ and $l, k = 1, \ldots, |Y|$
5:     $Q^X \leftarrow Q_X[\langle pa_{X\setminus Y} \rangle, x_i$
6:     $Q^Y \leftarrow Q_Y[\langle pa_{Y\setminus X} \rangle, y_k$
7: if $i = j \land k = l$
8:     $q_{(i,j),(k,l)}^{XY} \leftarrow q_{i,j}^X + q_{k,l}^Y$
9: else if $i = j \land k \neq l$
10:     $q_{(i,j),(k,l)}^{XY} \leftarrow q_{k,l}^Y$
11: else if $i \neq j \land k = l$
12:     $q_{(i,j),(k,l)}^{XY} \leftarrow q_{i,j}^X$
13: end if
14: end for
15: $Q_{XY|\langle pa_{XY} \rangle} \leftarrow Q^{XY}$
16: $Q_{XY|Pa(XY)} \leftarrow Q_{XY|Pa(XY)} \cup \{ Q_{XY|\langle pa_{XY} \rangle} \}$
17: end for
18: return $Q_{XY|Pa(XY)}$
Implementations

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

- CIM: 210 lines
- CTBN
- Inférence
  - Simple
  - Sampling
Mainly : synchronization of 3 very different objects :
Mainly: synchronization of 3 very different objects:

- oriented graph
  (gum.DiGraph),
Mainly: synchronization of 3 very different objects:

- oriented graph (gum.DiGraph),
- Discrete random variables in a dictionary (gum.DiscreteVariable)
Mainly: synchronization of 3 very different objects:

- oriented graph (gum.DiGraph),
- Discrete random variables in a dictionary (gum.DiscreteVariable)
- CIMs in a dictionary.
Mainly : synchronization of 3 very different objects :

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

```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, NodeID]:
    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
Inférence simple
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

Introduction, introspection, illustration
Algorithm 2.1 Forward sample CTBN.

\texttt{ForwardSample}(\mathcal{N})
\begin{algorithmic}[1]
\State \textbf{for each} \( X \in \mathcal{N} \)
\State \quad \text{choose} \( X(0) \) \text{ by sampling from} \mathcal{B} \\
\State \textbf{end for}
\State \( t \leftarrow 0 \), \( \sigma \leftarrow \emptyset \)
\State \textbf{repeat} until termination
\State \quad \text{Append}(\sigma, (X, t))
\State \quad \textbf{for each} \( X \in \mathcal{N} \)
\State \quad \quad \textbf{if} \ time(X) \neq \text{null} \textbf{ then} \text{ continue} \textbf{ end if}
\State \quad \quad \texttt{A}_X \leftarrow \texttt{A}_{X'|\texttt{Pa}(X)}
\State \quad \quad \texttt{i} \leftarrow X(t)
\State \quad \quad \Delta t \sim \text{Exponential}(a_{i,i})
\State \quad \quad \text{time}(X) \leftarrow t + \Delta t
\State \textbf{end for}
\State \quad \texttt{X}' \leftarrow \text{argmin}_{X \in \mathcal{N}} \text{time}(X)
\State \quad \texttt{t} \leftarrow \text{time}(\texttt{X}')
\State \quad \texttt{X}(t) \sim \text{Multinomial}(\texttt{A}_{\texttt{X}'}, \texttt{X}(t))
\State \quad \text{time}(\texttt{X}') \leftarrow \text{null}
\State \quad \textbf{for each} \( Y \in \text{Ch}(\texttt{X}') \)
\State \quad \quad \text{time}(Y) \leftarrow \text{null}
\State \textbf{end for}
\State \textbf{end repeat}
\State \textbf{return} \( \sigma \)
\end{algorithmic}
Forward Sampling

Algorithm 2.1 Forward sample CTBN.

\[\text{ForwardSample}(\mathcal{N})\]
1: for each \(X \in \mathcal{N}\)
2: \hspace{1em} choose \(X(0)\) by sampling from \(\mathcal{B}\)
3: end for
4: \(t \leftarrow 0\), \(\sigma \leftarrow \emptyset\)
5: repeat until termination
6: \hspace{1em} Append(\(\sigma, (X, t)\))
7: \hspace{1em} for each \(X \in \mathcal{N}\)
8: \hspace{2em} if \(\text{time}(X) \neq \text{null}\) then continue end for
9: \hspace{2em} \(A_X \leftarrow A_{X|\text{Pa}(X)}\)
10: \hspace{2em} \(i \leftarrow X(t)\)
11: \hspace{2em} \(\Delta t \sim \text{Exponential}(a_{i,i})\)
12: \hspace{2em} \(\text{time}(X) \leftarrow t + \Delta t\)
13: end for
14: \(X' \leftarrow \arg\min_{X \in \mathcal{N}} (\text{time}(X))\)
15: \(t \leftarrow \text{time}(X')\)
16: \(X(t) \sim \text{Multinomial}(A_{X'}, X(t))\)
17: \(\text{time}(X') = \text{null}\)
18: for each \(Y \in \text{Ch}(X')\)
19: \hspace{1em} \(\text{time}(Y) = \text{null}\)
20: end for
21: end repeat
22: return \(\sigma\)
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.
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 responsibilities
  - Interaction
    - gitlab issues, discord, gitter, linkedin, researchGate, what else?
  - Structuration
    - community (?), consortium (?)
  - Scientific orientation?
    - models
    - algorithms
    - scientific committee
    - ?

- Development orientation?
  - weaknesses, strengths
  - missing features
  - Ragrum, JSagrum
  - Steering committee
  - ?