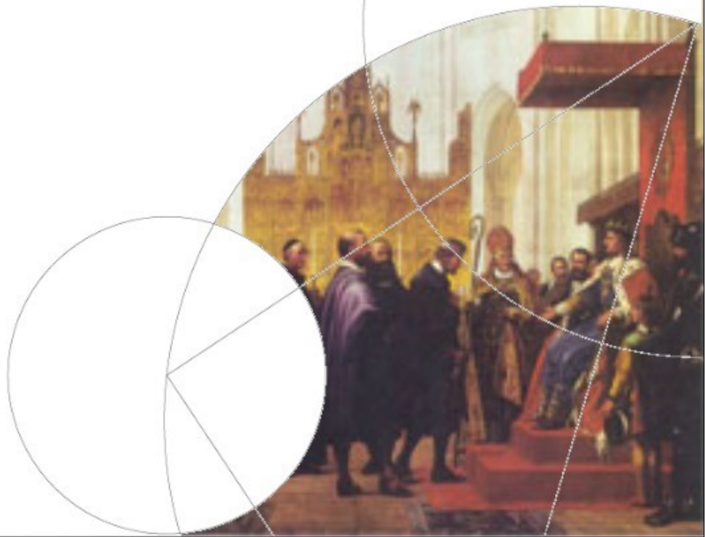
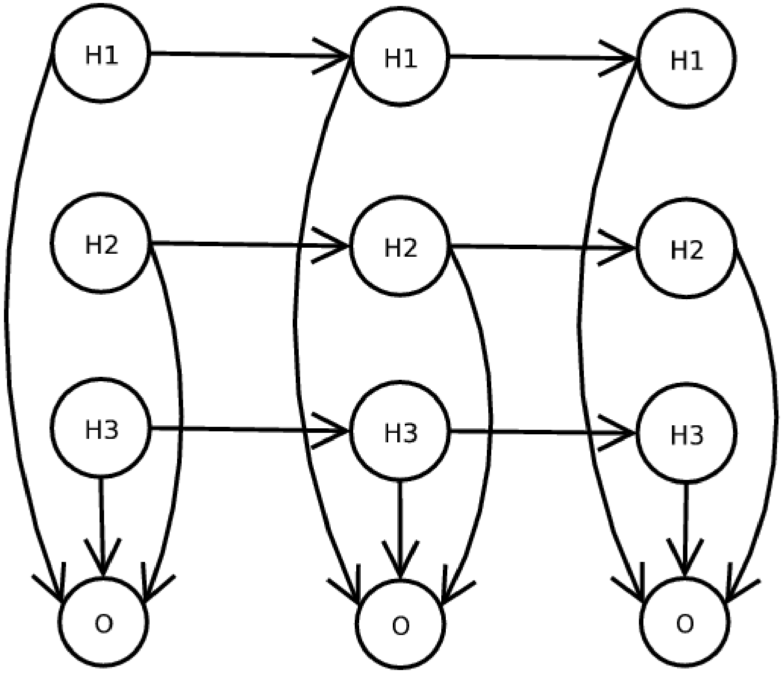
Mocapy++

implementation of a
mixed discrete/continuous node



Mocapy: Graphical Models: Dynamic Bayesian Networks

Mocapy: Dynamic Bayesian Networks topology



- Directed Acyclic Graphs
  - nodes are probability distributions
  - arcs encode conditional independences
  - all the slices are equal
  - first order dependences between slides
- Topological independences → computational efficiency

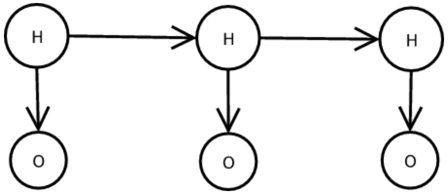
First implementation: **Mo**nte **Ca**rlo **Py**thon

- now faster implementation in C++: Mocapy++ (same architecture)

Your task: to extend Mocapy++, defining a new node type

- Maximum likelihood estimation
- Density calculation
- Sampling
- Technicality: data storage: save/load data, and save/restore models

Mocapy++: background overview



Mocapy: Inference

identify the distribution of hidden nodes **h**
given the model and the observed quantities

$$P(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta})$$

Mocapy: Learning

estimate the parameters of the model

approximate Expectation-Maximization:
iterative Maximum Likelihood while (Markov) blanket sampling

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{\mathbf{h}} P(\mathbf{h}, \mathbf{d} \mid \boldsymbol{\theta})$$

$$\boldsymbol{\theta}_{\mathrm{new}} = \arg\max_{\boldsymbol{\theta}} \sum_{\mathbf{h}} P(\mathbf{h} \mid \mathbf{d}, \boldsymbol{\theta}_{\mathrm{old}}) \log P(\mathbf{h}, \mathbf{d} \mid \boldsymbol{\theta})$$

in practice: call the Stochastic-EM functionalities, in the Engine section of the framework

Mocapy++: framework usage

Your test case should mimic *examples/hmm_simple.cpp*

Nodes: definitions, parameters

DBN: topology definition

Sampling algorithm: Gibbs MC

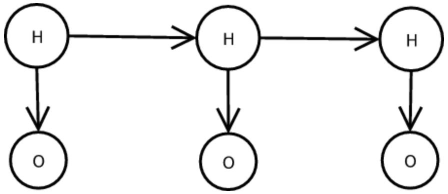
EM engine: sampling + DBN object (Stochastic-EM)

load (in this order):

- mismask: label for the node values (**mis**sing **mask**)
  - 0: MOCAPY_OBSERVED
  - 1: MOCAPY_HIDDEN
  - 2: MOCAPY_MISSING
- data: one value per dimension,
  in the order specified by the DBN topology

perform the learning (Stochastic-EM)

print the learned nodes

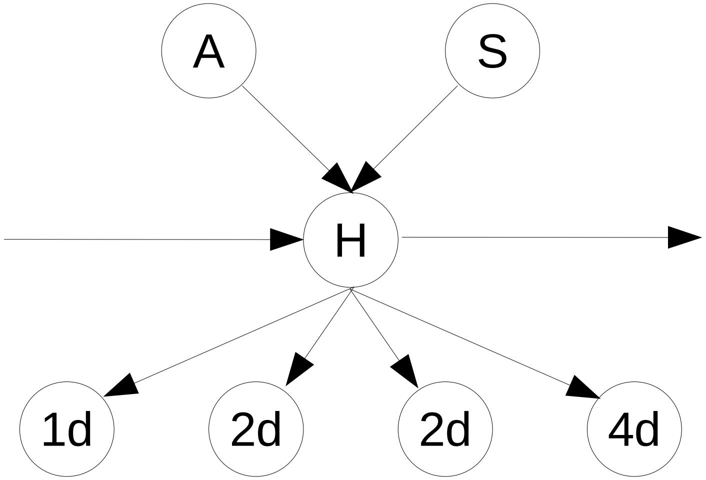


```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

#include "mocapy.h"
using namespace mocapy;
using namespace std;

int main(void) {
        // The dynamic Bayesian network
        DBN dbn;

        // Nodes in slice 1
        Node* h1 = NodeFactory::new_discrete_node(5, "h1");
        Node* o1 = NodeFactory::new_discrete_node(2, "o1");

        // Nodes in slice 2
        Node* h2 = NodeFactory::new_discrete_node(5, "h2");

        // Set architecture
        dbn.set_slices(vec(h1, o1), vec(h2, o1));

        dbn.add_intra("h1", "o1");
        dbn.add_inter("h1", "h2");
        dbn.construct();

        cout << "Loading traindata" << endl;
        GibbsRandom mcmc = GibbsRandom(&dbn);

        EMEngine em = EMEngine(&dbn, &mcmc);
        em.load_mismask("data/mismask.dat");
        em.load_weights("data/weights.dat");
        em.load_sequences("data/traindata.dat");

        cout << "Starting EM loop" << endl;
        for (uint i=0; i<100; i++) {
                em.do_E_step(20, 10);
                double ll = em.get_loglik();
                cout << "LL= " << ll << endl;
                em.do_M_step();
        }

        cout << "h1: " << *h1 << endl;
        cout << "o1: " << *o1 << endl;
        cout << "h2: " << *h2 << endl;

        return EXIT_SUCCESS;
}
```

Mocapy++: datafiles

Mocapy: format for datafiles

```
// File format:
// A line contains values in a slice
//       separated by "sep" (default space).
// A block of lines therefore represents a sequence.
// An empty line separates two sequences.
// Lines starting with # are ignored
```



input.mismask (order: has to be consistent with topology in .cpp file):

```
 0 0  1       0                 0                 0                                   0
 0 0  1       0                 0                 0                                   0
 0 0  1       0                 0                 0                                   0
 0 0  1       0                 0                 0                                   0
```

input.data (order: consistent with mismask):

```
 8 3  0  0.4286  -4.2688  -2.6449  -2.4226  -2.3128  -0.9675  -0.7078  -0.8320  -0.6090
 8 2  0  0.5385  -4.4054  -2.4140  -2.2166  -1.6194  -3.8429  -1.6061  -1.8543  -0.9892
17 2  0  0.3750  -4.4473  -2.3684  -2.6483  -1.8028  -4.5492  -2.6714  -2.5935  -2.3782
 8 2  0  0.3636  -4.5259  -2.3819  -2.2348  -1.5630  -4.6230  -2.7179  -2.5563  -1.9557
```

Mocapy++: download, compilation

Mocapy++: download from sourceforge.net (anonymous svn)

```
svn co https://mocapy.svn.sourceforge.net/svnroot/mocapy/Mocapy .
```

Compilation: cmake. Your test program can be compiled

as an external program:

```
<include mocapy.h>
link with: mocapy++, lapack, boost.serialization libraries
```

as an example in the examples/ directory (easier):

place your code in the examples/ directory
edit the CMakeLists.txt

```
SET(PROGS ...)
```

Manual: file /MANUAL.pdf in the root of the repository

ch. 2: installation
ch. 1,3: theoretical foundations
ch. 4: I/O, general usage
ch. 6, 7: design, extension

Mocapy++: your task: overview

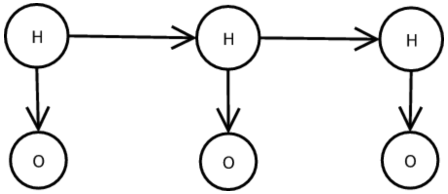
Your test case should mimic *examples/hmm_simple.cpp*

input.mismask:

```
1    0
1    0
1    0
```

input.data (here 2 sequences):

```
# first column: hidden node
# first dimension: 0: indicator for discrete
# first dimension: 1: indicator for continuous
0   0   3
0   0   2
0   1   0
0   0   0
0   1   0.5
0   1   3.75

0   0   3
0   0   1
# errors: skip them, but warn the user
0   0   3.75
0   0  -5
```



```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

#include "mocapy.h"
using namespace mocapy;
using namespace std;

int main(void) {
        // The dynamic Bayesian network
        DBN dbn;

        // Nodes in slice 1
        Node* h1 = NodeFactory::new_discrete_node(5, "h1");
        Node* o1 = NodeFactory::new_discrete_node(2, "o1");

        // Nodes in slice 2
        Node* h2 = NodeFactory::new_discrete_node(5, "h2");

        // Set architecture
        dbn.set_slices(vec(h1, o1), vec(h2, o1));

        dbn.add_intra("h1", "o1");
        dbn.add_inter("h1", "h2");
        dbn.construct();

        cout << "Loading traindata" << endl;
        GibbsRandom mcmc = GibbsRandom(&dbn);

        EMEngine em = EMEngine(&dbn, &mcmc);
        em.load_mismask("data/mismask.dat");
        ~~em.load_weights("data/weights.dat");~~
        em.load_sequences("data/traindata.dat");

        cout << "Starting EM loop" << endl;
        for (uint i=0; i<100; i++) {
                em.do_E_step(20, 10);
                double ll = em.get_loglik();
                cout << "LL= " << ll << endl;
                em.do_M_step();
        }

        cout << "h1: " << *h1 << endl;
        cout << "o1: " << *o1 << endl;
        cout << "h2: " << *h2 << endl;

        return EXIT_SUCCESS;
}
```

Mocapy++: your task

Mocapy: your task: high level

- Input: categorical and continuous
  - algorithms from DiscreteNode class. Continuous estimation: extract code from GaussianNode (skip the multidimensional and shrinkage case)
- Define a 2d node
  - first dimension: indicator (0: discrete, 1: continuous)
  - second dimension: value of the node
  - this node requires exactly one discrete parent
- Learning: 2 steps (to do for each value of the single discrete parent):
  - learn the ratio between categorical and continuous output types
    - simply count the number of categorical, divided by the number of valid observations
    - learn the parameters for the two different sets of value types, independently
    - we split the input dataset in categorical dataset and Gaussian dataset
- Sampling: 2 steps
  - sample a random number (a percentage)
  - lower or equal to categorical ratio threshold? Sample from the DiscreteNode algorithms
  - higher? Sample from GaussianNode algorithms

Mocapy++: architecture

Mocapy++: framework and changes

Engine: you simply need to use it

as you see, the node object is
the glue between the engine and
the distribution frameworks

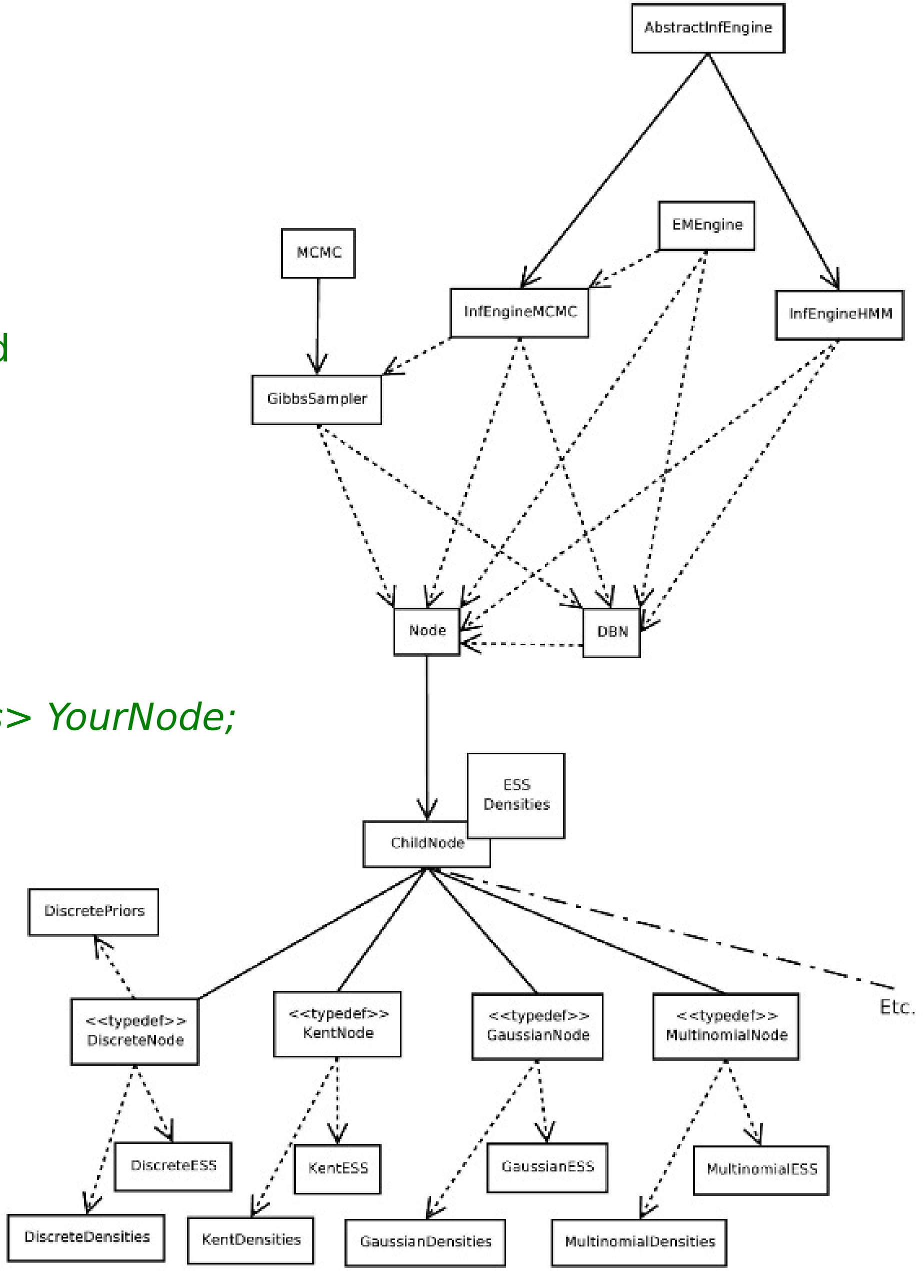
Node → ChildNode template

template Childnode<xESS, xDensities> YourNode;

Node Internals

ESS: Expected Sufficient Stats
add/store sample
for hidden node operations

densities: estimation, sampling



Mocapy++: your task: details

Mocapy: your task: part of the framework to change and deliver

- src/framework/dbn.h:
  - include lines
  - typedef ChildNode<YourESS, YourDensities> YourNode;

- src/yournode/youress.{h,cpp}
  - YourESS class, inherits from abstract ESSBase. Override:
    - construct(): define the appropriate shape of the ESS
    - add_ptv(): add a sample point to the ESS

- src/yournode/yourdensities.{h,cpp}
  - YourDensities class, inherits from abstract DensitiesBase. Override:
    - construct(parent_sizes): initialize parameters (mean, covariance, CPD, etc.).
    - estimate(ess): estimate the parameters of the node based on the ESS
    - sample(ptv): return a sample based on the parent values
    - get_lik(ptv, log): return the likelihood P(child|parent)
    - get_parameters(): return the parameters of the distribution

- examples/yournode_hmm.cpp
  - as per simple_hmm.cpp (edit the CMakeLists.txt)

- make sure the dbn.save() and dbn.load() methods work

Mocapy++: your addenda

Mocapy: simple example on how to wrap the two nodes

Let's suppose the categorical output has probability 0.3

sampling

- Sample the threshold. Is it lower than 0.3?
  - Sample from the categorical routines
  - Else, sample from the Gaussian routines

Learning

- Discrete set: discrete/discretedensities.*, discrete/discreteess.*
- Continuous set: gaussian/*
    - start: examples/hmm_gauss_1d.cpp
  - public functions: construct(), estimate(), get_lik()
  - private functions: calc_means(), calc_cov_full() (copy-and-paste is the easier option)

Likelihood estimation for a given parent value

- Note: θ is the probability vector of the discrete distribution for that parent value
- Note: G is Gaussian with mean and std. dev. μ,σ for that parent value
- P(categorical, 2)=P(2 |θ)*P(categorical) ← 0.3
- P(continuous, 3.5)=G(3.5 | μ,σ)*P(continuous) ← 0.7

Details for the instantiation of the nodes: src/framework/nodefactory.*, examples/*.cpp

Mocapy++: overview of the task

Mocapy++: task

download
compile

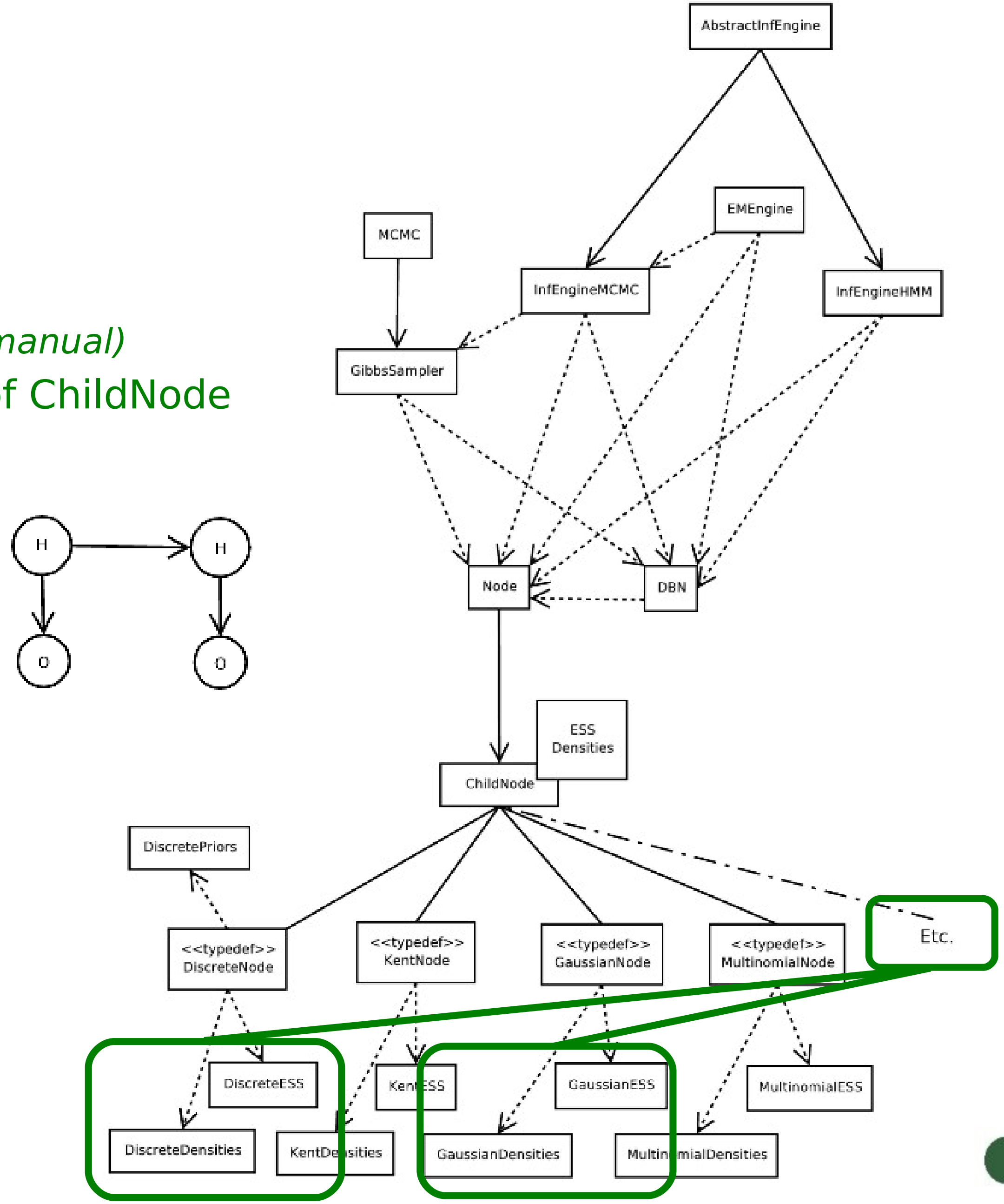
write ESS, Densities *(mocapy++ manual)*
define node as a specialization of ChildNode

load data file, mismask

write your test program

Deliver: tar.gz archive

report
your code (test included)
your data files


H
H
O
O
AbstractInfEngine
MCMC
EMEngine
InfEngineMCMC
InfEngineHMM
GibbsSampler
Node
DBN
ESS
Densities
ChildNode
DiscretePriors
<<typedef>>
DiscreteNode
<<typedef>>
KentNode
<<typedef>>
GaussianNode
<<typedef>>
MultinomialNode
Etc.
DiscreteESS
KentESS
GaussianESS
MultinomialESS
DiscreteDensities
KentDensities
GaussianDensities
MultinomialDensities