


# BNJ 2.03a Intermediate Developer Tutorial

**Roby Joehanes**  
(revised by William H. Hsu)  
Kansas State University  
KDD Laboratory  
<http://www.kddresearch.org>  
<http://bndev.sourceforge.net>



## Contents

- **Introduction**
- Core Classes
- Inference Classes
- Data Classes
- Learning Classes
- Converter Classes
- Genetic Algorithm Classes

<http://bndev.sourceforge.net>

## BNJ 2.0: What's New

- Total revamp from 1.0
- Aimed towards stability, flexibility, maintainability, and speed
- New functionality added
- Status: alpha
  - API not finalized yet
  - More mature on learning side than inference

<http://bndev.sourceforge.net>

## Core Architecture

- Divided into several parts:
  - Core classes – **edu.ksu.cis.bnj.bbn** package  
For expressing graphs, nodes, edges, cpts (CPF's), PDFs (cpt's entries).
  - Inference – **edu.ksu.cis.bnj.bbn.inference** package  
Exact and inexact inference
  - Learning – **edu.ksu.cis.bnj.bbn.learning**  
Structure learning
  - PRM – **edu.ksu.cis.bnj.bbn.prm**  
All classes necessary for probabilistic relational models
  - Data – **edu.ksu.cis.kdd.data**  
Representing data for learning methods

<http://bndev.sourceforge.net>

# Auxiliary Architecture

- Converter – **edu.ksu.cis.bnj.bbn.converter**  
Loading, saving, converting network files
- Data converter – **edu.ksu.cis.kdd.data.converter**  
Loading, saving, converting data files
- GUI – **edu.ksu.cis.bnj.gui**  
Graphical user interface components
- Genetic algorithm – **edu.ksu.cis.kdd.ga**  
Mainly used for GAWK (*GA Wrapper for K2*)
- Bayes classifier – **edu.ksu.cis.kdd.classifier**  
Now slightly updated – Machine Learning (Fall, 2001)  
class project, Joehanes
- Other utilities

<http://bndev.sourceforge.net>

# Contents

- Introduction
- **Core Classes**
- Inference Classes
- Data Classes
- Learning Classes
- Converter Classes
- Genetic Algorithm Classes

<http://bndev.sourceforge.net>

## Core Classes [1]

- Mainly inherited from OpenJGraph
- **BBNGraph**: Bayes Net Graph
- **BBNNode**: Bayes Net Node
- **BBNCPT**: Conditional Probability Table
  - Called CPF[unction] because support for continuous values planned
  - Full CPFs not yet implemented
- **BBNPDF**: CPT entries
  - PDF = Probabilistic Distribution Function; again, for continuous values

<http://bndev.sourceforge.net>

## Core Classes [2]

- **BBNValue**: Abstract class for discrete (**BBNDiscreteValue**) and continuous values (**BBNContinuousValue**) → will be phased out in upcoming release
- **BBNConstant**: Derived class of **BBNPDF** for representing constants (superfluous, will be phased out)
- **EvidenceParser** → Class to parse evidence values. (should not be called externally)

<http://bndev.sourceforge.net>

## BBNGraph [1]

- To create a Bayes net graph:  
`BBNGraph g = new BBNGraph();`
- To load a Bayes net:  
`BBNGraph g = BBNGraph.load("file");`
- To save a graph:  
`g.save("file");` – or –  
`g.save("file", "format");`  
*e.g.*, `g.save("mynet.net", "net");`
- To load evidence:  
`g.loadEvidence("evidencefile");`

<http://bndev.sourceforge.net>

## BBNGraph [2]

- To save evidence:  
`g.saveEvidence("evidencefile");`
- To print graph contents for debugging:  
`System.out.println(g.toString());`
- To add or remove a node:  
`g.addNode(bbnNode);`  
`g.removeNode(bbnNode);` // related edges will also be deleted
- To add or remove an edge:  
`g.addEdge(node1, node2);`  
`g.removeEdge(node1, node2);`
- To topologically sort a graph:  
`List nodeList = g.topologicalSort();`

<http://bndev.sourceforge.net>

## BBNNode [1]

- To create a node:  
`BBNNode node = new BBNNode();`  
`node.setName("name");`  
`// must then add node to graph`
- To set node values:  
`BBNDiscreteValue v = new`  
`BBNDiscreteValue();`  
`v.add("v1"); v.add("v2"); // ... and so on`  
`node.setValues(v);`
- To get node values:  
`BBNValue v = node.getValues();`
- To set or get evidence:  
`node.setEvidenceValue(val); // val must be in the`  
`BBNValue`  
`Object val = node.getEvidenceValue();`

<http://bndev.sourceforge.net>

## BBNNode [2]

- To turn node into Decision / Utility node:  
`node.setType(BBNNode.DECISION);`  
`node.setType(BBNNode.UTILITY);`
- To inquire about a node:  
`node.isDecision();`  
`node.isUtility();`  
`node.isEvidence();`
- To access CPT:  
`BBNCPF cpf = node.getCPF();`  
`// Note: Don't use query from the node directly`  
`// as it will be phased out soon`

<http://bndev.sourceforge.net>

## BBNCPF (This will be phased out)

- CPFs should not be created individually unless you know what you are doing (e.g. creating ICPTs for SIS or AIS)
- To create a CPF object yourself, use:

```
List l = new LinkedList();
l.add("node1"); l.add("node2");
// etc, add the node names involved
for the CPT (in string)
BBNCPF cpf = new BBNCPF(l);
// Note: This will be obsolete soon.
```

<http://bndev.sourceforge.net>

## Querying BBNCPF [1]

- To query CPF: construct hash table
- Example
  - Let  $a$  and  $b$  be the parents of node  $c$
  - Let all nodes be Boolean
  - To query content of CPT entry for ( $a = \text{true}$ ,  $b = \text{true}$ ,  $c = \text{false}$ ):

```
Hashtable t = new Hashtable();
t.put("a", "true"); t.put("b", "true");
t.put("c", "false");
double value = c.query(t);
```
- Note: BBNCPF to become obsolete (v2.2b, v2.3)

<http://bndev.sourceforge.net>

## Querying BBNCPPF [2]

- The code:

```
Hashtable t = new Hashtable();
t.put("a", "true"); t.put("c",
"false");
double value = c.query(t);
```

will result in:

```
value = c.query(t U "b = true") + c.query(t U
"b = false");
```
- So, if we omit *b* from the hash table, we are effectively marginalizing on *b* from *c*'s table.

<http://bndev.sourceforge.net>

## Reason for CPF Overhaul

- **Q: Why do we need to revamp CPFs?**  
A: CPFs are the performance hog in BNJ. They admit an  $\exp^{\exp(N)}$  space requirement and will thus cause thrashing when computing large networks.
- Originally used because space was thought to be less of an issue than speed
- Revamp is underway
- Bart Peintner has also provided a solution

<http://bndev.sourceforge.net>

# Contents

- Introduction
- Core Classes
- **Inference Classes**
- Data Classes
- Learning Classes
- Converter Classes
- Genetic Algorithm Classes

<http://bndev.sourceforge.net>

# Inference Classes

- **Inference**: Abstract class that all inference modules should inherit from
- **ExactInference**: Abstract class for all exact inference
- **ApproximateInference**: Abstract class for approximate inference
- **MCMC**: For MCMC based approximate inference (**`edu.inference.approximate.sampling`**)

<http://bndev.sourceforge.net>

## Invoking Available Inference Classes

- Example (Lauritzen-Spiegelhalter, *i.e.*, junction tree):  

```
BBNGraph g = BBNGraph.load("netfile");  
g.loadEvidence("evidenceFile"); // if  
needed  
LS ls = new LS(g);  
InferenceResult result =  
ls.getMarginals();
```
- Variable result: printable hash table  

```
System.out.println(result.toString());
```
- Other inference classes are invoked in the same way.
- Note: actual inference happens in `getMarginals()`
- May want to set some options before it

<http://bndev.sourceforge.net>

## Getting the Most Probable Explanaton (MPE)

- BNJ inference provides default method for getting MPE
  - `Hashtable getMPE()`
  - Returns a hash table of (node name → mpe value)
- BNJ does not provide a customized MPE routine for each inference, so it is very slow (*i.e.*, does it naively).
- MAP: not done yet

<http://bndev.sourceforge.net>

## Available Inference Algorithms

- LS / Junction tree
- *ElimBel* (Bucket elimination)
- Pearl Tree propagation (currently buggy)
- Forward Sampling
- Logic Sampling
- Likelihood Weighting
- Self-Importance Sampling
- Adaptive Important Sampling
- Cutset and Bounded Cutset (buggy)
- Chavez MCMC (buggy)
- Pearl MCMC (buggy)

<http://bndev.sourceforge.net>

## Customizing Inference Classes

- Ideal case: plug-in system
  - User can build own inference modules
  - “Just works” with BNJ
- Main infrastructure relies on Java Reflection API extended on file `FileClassLoader.java`
- User must inherit from
  - `ExactInference` for exact inference methods
  - `ApproximateInference` for inexact ones
- Must inherit at least `getMarginals()` method

<http://bndev.sourceforge.net>

# Customizing Inference Classes Example

```
public class MyInference extends ExactInference {  
    // or extends ApproximateInference  
    public MyInference(BBNGraph g) {  
        // Your constructor  
    }  
    public InferenceResult getMarginals() {  
        // write your inference routine here  
    }  
}
```

- Next, add inference class to your Java classpath
- Modify config.xml to make BNJ GUI see your inference class (not done yet)

<http://bndev.sourceforge.net>

# Contents

- Introduction
- Core Classes
- Inference Classes
- **Data Classes**
- Learning Classes
- Converter Classes
- Genetic Algorithm Classes

<http://bndev.sourceforge.net>

## Data Classes [1]

- For structure learning
- Encapsulate data
- May be local or remote
- Loading data (locally):  

```
Database db = Database.load("filename");
```

  
or  

```
Database db = Database.load("file",  
"format");
```

<http://bndev.sourceforge.net>

## Data Classes [2]

- Database may contain multiple tables
  - Probabilistic Relational Models (PRMs)
  - Other Relational Graphical Models (RGMs)
- To check  

```
List tables = db.getTables();  
if (tables.size() > 1)  
    // PRM or other RGM  
else  
    // single table (traditional BN)
```

<http://bndev.sourceforge.net>

## Remote Database Connection [1]: Loading

- Loading data remotely is slightly different:

```
Class.forName(driver);  
Connection c = DriverManager.getConnection(url, login,  
passwd);  
Database db = Database.importRemoteSchema(c);
```

- Driver – one of the following

<code>sun.jdbc.odbc.JdbcOdbcDriver</code>	ODBC-based driver, e.g., MS Access
<code>org.gjt.mm.mysql.Driver</code>	mySQL
<code>oracle.jdbc.driver.OracleDriver</code>	ORACLE
<code>org.postgresql.Driver</code>	PostgreSQL

<http://bndev.sourceforge.net>

## Remote Database Connection: Example

```
String driver = "org.gjt.mm.mysql.Driver";  
String url = "jdbc:mysql://localhost/mydb";  
String login = "mylogon", passwd = "opensesame";  
// Copy and paste the connection code in the  
// previous slide
```

For ORACLE, the URL is slightly different

```
url = "jdbc:oracle:thin:@localhost:1521:mydb";
```

<http://bndev.sourceforge.net>

## Database API [1]

- Getting all available attributes:

```
List l = db.getAttributes();
```

- Getting satellite attributes (non-primary key and non-reference key):

```
List l = db.getRelevantAttributes();
```

```
// In case of single table, getAttributes ==  
getRelevantAttributes()
```

<http://bndev.sourceforge.net>

## Database API [2]

- Getting all available tuples:

```
List l = db.getTuples();
```

```
// This will import all tuples from remote to  
local for remote connection
```

- Subsampling (returns  $n$  random tuples from current data):

```
Data d = db.subsample(n);
```

```
// Currently works only if db is single table
```

<http://bndev.sourceforge.net>

## Database API [3]

- Getting and setting weights:  

```
double[] weights = db.getWeights();  
db.setWeights(weights);
```
- Getting tallyer (class for counting items):  

```
Tally tallyer = db.getTallyer();
```

<http://bndev.sourceforge.net>

## Tallyer [1]

- Crucial component for data counting used in structure learning
- Contains many caches to speed up calculations
- String attributes and values are converted to integers. So, watch out!
- `Tallyer` may be filtered recursively using `createSubTally` method in order for efficiency
- Final tally counted using `size()` method

<http://bndev.sourceforge.net>

## Tallyer [2]

- To get size of tally (number of tuples that match criterion):

```
int size = t.getSize();
```

- Filter out tallyer that matches criterion:

```
Tally subTally = t.createSubTally(1, 2);
```

This means that we filter out the tuples whose attribute #1 doesn't have value 2

<http://bndev.sourceforge.net>

## Tallyer [3]

- `tally` method is to count the number of tuples that satisfies the criterion:

```
int n = t.tally(1, 2);
```

// n holds the number of tuples whose attribute #1 contains value 2

- i.e., `tally` is like invoking `createSubTally()` and then `size()`

<http://bndev.sourceforge.net>

## Tallyer [4]

- `createSubTally(int[], int[])` and `tally(int[], int[])` are similar to the single attribute ones
- `getUnderlyingData()` gets `Data` object the tallyer is associated with
- `getRelevantAttributeIndices()` gets satellite attribute indices (non-primary key and non-reference key), in `int[]`

<http://bndev.sourceforge.net>

## Tallyer [5]

- `groupedTally(int[])` is for getting the counts for all possible combinations of assignments to the indices
- Example
  - Suppose we need to query attribute #0, 1, and 2 and each has value 0 and 1
  - Grouped tally of [0,1,2] would return the list of the counts of [0=0,1=0,2=0], [0=0,1=0,2=1], ... , [0=1,1=1,2=1]
- Very handy for PRM structural learning

<http://bndev.sourceforge.net>

## Importing Remote Databases

- Note: it is possible to have the remote database imported locally
- However, local database tallyer is currently very buggy
- If you want to fix it...
  - It is in the `LocalDatabaseTally`
  - Bug is in join code

<http://bndev.sourceforge.net>

## Contents

- Introduction
- Core Classes
- Inference Classes
- Data Classes
- **Learning Classes**
- Converter Classes
- Genetic Algorithm Classes

<http://bndev.sourceforge.net>

## Learning Classes

- Just like inference classes, `Learning` has a parent class `Learner` from which all structural learning classes should inherit
- `ScoreBasedLearner` – parent class for learner that has scores in it
- `CIBasedLearner` – parent class for conditional-independence-based learner
- Currently BNJ only has `ScoreBasedLearner`

<http://bndev.sourceforge.net>

## Invoking Learning Module

- Very similar to invoking inference classes:

```
// Load database as shown in previous slides
```

```
K2 k2 = new K2(db);
```

```
BBNGraph g = k2.getGraph();
```

```
// g is the learned graph
```

<http://bndev.sourceforge.net>

## Available Learning Methods

- K2
- Greedy structure learning
- Hill-climbing
- Adversarial hill-climbing
- Simulated annealing
- GAWK (Genetic Algorithm Wrapper for K2)
- Note: for PRMs, we only retrofit K2, not the others.

<http://bndev.sourceforge.net>

## Customizing Learning Classes

- Must inherit from `scoreBasedLearner` at present
- Also a plug-in style (not yet mature)
- Must implement `getGraph()` method

<http://bndev.sourceforge.net>

# Learning Class Skeleton

## ■ Example skeleton:

```
public class MyLearner extends ScoreBasedLearner {
    public MyLearner(Data d) { super(d); } // only data
    // Data and candidate scorer
    public MyLearner(Data d, LearnerScore s) { super(d,s); }
    // Data, candidate scorer and structure scorer
    public MyLearner(Data d, LearnerScore s1, LearnerScore s2)
    { super(d,s1,s2); }
    public BBNGraph getGraph() {
        // Your learning algorithm is here
    }
}
```

<http://bndev.sourceforge.net>

# ScoreBasedLearner

- Must have a scorer to evaluate possible
  - Candidates (`candidateScorer`)
  - Structures (`structureScorer`)
- We may need one of them or both.
  - K2 will need only `candidateScorer`
  - GreedySL should need only `structureScorer`  
(in the code it mistakenly uses `candidateScorer`)

<http://bndev.sourceforge.net>

## LearnerScore

- Needed for `scoreBasedLearner` to evaluate candidate/structure
- Idea: examine different scoring schemes to evaluate impact on learning algorithm
- Currently we have `BDEScore` and `DiscrepancyScore`
- Others are just skeletons at the moment

<http://bndev.sourceforge.net>

## Customizing LearnerScore [1]

- To make a new scoring scheme, must inherit `LearnerScore`
- Must override `double getScore(int curNode, int candidate, Set[] parentTable)`
- `curNode`: index of the node being evaluated
- `candidate`: index of candidate parent (-1 if not applicable; *i.e.*, for structure scorer)
- `parentTable`: set of integers of currently evolving structure.

<http://bndev.sourceforge.net>

## Customizing LearnerScore [2]

- In `LearnerScore`, children have access to `tallyer`
- This learner score expected to use `tallyer` extensively

<http://bndev.sourceforge.net>

## Contents

- Introduction
- Core Classes
- Inference Classes
- Data Classes
- Learning Classes
- **Converter Classes**
- Genetic Algorithm Classes

<http://bndev.sourceforge.net>

## Converter Classes

- Like `Inference` and `Learning`, all network converters implement parent interface: `Converter` (in **`edu.ksu.cis.bnj.bbn.converter`**)
- Loading/saving should not be invoked directly, but rather through `BBNGraph.load` Or `BBNGraph.save`
- Focuses on customizing converter classes

<http://bndev.sourceforge.net>

## Building Your Own Converter

- Must implement `Converter` interface
- Implement methods
  - `initialize()`
  - `BBNGraph load(InputStream)`
  - `save (OutputStream, BBNGraph)`
- Initialize method is called before `load()`, `save()`
- `load()`, `save()` methods: self-explanatory

<http://bndev.sourceforge.net>

## Testing Your Converter

- First, test your converter prior to BNJ integration
  - Implement own main method
  - Test it there by invoking load/save
- Second, put your converter class in Java class path

<http://bndev.sourceforge.net>

## Modify config.xml

- Inside `<CONVERTERS>` tag, add your own fields
- Example

```
<CONVERTER DESCRIPTION="My Fancy Bayes Net format"
EXTENSION="fbn" PACKAGENAME="mypackage.myconverter"
CLASSNAME="MyConverter" />
```
- Save config.xml
- BNJ should now recognize your converter

<http://bndev.sourceforge.net>

## Customizing Data Converter [1]

- Data converter works similarly to network converters
- Implements Converter interface
- Contained in package **edu.ksu.cis.kdd.data.converter**
- Contains 3 methods to inherit
  - `initialize()`
  - `Database load(InputStream)`
  - `save (OutputStream, Database)`

<http://bndev.sourceforge.net>

## Customizing Data Converter [2]

- First, test your converter prior to BNJ integration
- Second, modify config.xml under `<DATA CONVERTERS>` tag
- Example:

```
<DATA CONVERTER DESCRIPTION="My Fancy Data format" EXTENSION="fdf"
  PACKAGENAME=" mypackage.mydataconverter " CLASSNAME="MyDataConverter" />
```
- Save config.xml
- BNJ should recognize your data converter

<http://bndev.sourceforge.net>

# Contents

- Introduction
- Core Classes
- Inference Classes
- Data Classes
- Learning Classes
- Converter Classes
- **Genetic Algorithm Classes**

<http://bndev.sourceforge.net>

# Genetic Algorithm API in BNJ

- BNJ contains GA API because of GAWK module
- Simple but general enough for anyone to use
- Modeled after Evolutionary Computation in Java (ECJ) API  
<http://www.cs.umd.edu/projects/plus/ec/ecj>

<http://bndev.sourceforge.net>

## Population

- Contains of several sub-populations (demes)
- Each sub-population contains possibly many GA individuals

<http://bndev.sourceforge.net>

## Chromosome

- Each individual must inherit `Chromosome` class
- `Chromosome` must override methods
  - `clone()`: clone itself
  - `equals(Object)`: to test equality
  - `createObject()`: create fresh, blank object
  - `toString()`: for debugging (optional)

<http://bndev.sourceforge.net>

# Operators

- Also flexible
- Can create own operators, other than existing `CrossOverOp` and `MutationOp`
- Must inherit
  - constructor that specifies number of operands
  - `Chromosome apply(Chromosome[] ind)`
- Input: array of individuals that act as operands
- Must return new individual as result of applying operator
- See `shuffleOp` as an example
  - Simple shuffling operator
  - Needs only 1 operand

<http://bndev.sourceforge.net>

# Fitness Function

- Must have a fitness function to examine fitness score of GA individual
- How
  - Create a class that implements Fitness interface
  - Need to override method `double getFitness(Chromosome)`
- Input: individual to be examined
- Output: fitness score

<http://bndev.sourceforge.net>

## How to Use Them

```
// create a population
pop = new Population();
// only has 1 sub population
pop.subpop = new Subpopulation[1];

// instantiate your fitness function. It must
// implement Fitness interface
fitnessFunction = new MyFitnessFunction();

// create the sub-population with the subPopulationSize
// an instance of the individual, and the fitness function
pop.subpop[0] = new Subpopulation(subPopulationSize, new
    MyChrom(attributeSize), fitnessFunction);

// Adding operators. Here, it means that CrossOver is done
// 0.8 times and mutation is done 0.2 times.
pop.subpop[0].addOperator(new CrossOverOp(), 0.8);
pop.subpop[0].addOperator(new MutationOp(), 0.2);

// Evolve the GA for numGenerations times
pop.evolve(numGenerations);
```

<http://bndev.sourceforge.net>

## Note on Genetic Algorithms in BNJ

- Current implementation of the GA is very simple
- No multi-threaded (distributed) version yet

<http://bndev.sourceforge.net>