

SMILE Wrappers

Programmer's Manual

Version 2.4.R1, Built on 10/15/2025
BayesFusion, LLC

This page is intentionally left blank.

1. Introduction	9
2. Licensing	11
3. What's new in SMILE 2	13
4. Platforms and wrappers	15
4.1 Python and PySMILE	16
4.2 Java and jSMILE	17
4.2.1 Maven	17
4.3 .NET and SMILE.NET	19
4.4 R and rSMILE	20
5. Hello, SMILE Wrapper!	21
5.1 Success/Forecast model	22
5.2 VentureBN.xdsl	22
5.3 The program	23
5.3.1 Hello.py	24
5.3.2 Hello.java	25
5.3.3 Hello.cs	25
5.3.4 Hello.R	25
6. Using SMILE Wrappers	27
6.1 Error handling	28
6.2 Identifiers	28
6.3 Networks, nodes and arcs	28
6.3.1 Network	29
6.3.2 Nodes	29
6.3.2.1 Node types	29
6.3.2.2 Creating and deleting nodes	30
6.3.2.3 Iterating over nodes	30
6.3.2.4 Node definition	31
6.3.2.5 Node value	33
6.3.2.6 Node evidence	34
6.3.2.7 Other node attributes	36
6.3.3 Arcs	38
6.4 Multidimensional arrays	39
6.5 Input and output	40
6.6 Inference	41
6.6.1 Target nodes	42

6.6.2	Noisy-MAX decomposition	43
6.6.3	Probability of evidence	43
6.6.4	Annealed MAP	44
6.6.5	Most probable explanation	44
6.7	Case manager	45
6.8	User properties	45
6.9	Submodels	45
6.10	Discrete nodes and numeric domains	48
6.10.1	Outcome intervals	48
6.10.2	Outcome point values	49
6.11	Canonical nodes	50
6.11.1	Noisy-MAX	51
6.11.2	Noisy-Adder	53
6.12	Influence diagrams	54
6.13	Dynamic Bayesian networks	54
6.13.1	Temporal node types	55
6.13.2	Temporal arcs	56
6.13.3	Unrolling	57
6.13.4	Temporal definitions	59
6.13.5	Temporal evidence	60
6.13.6	Temporal beliefs	62
6.14	Continuous models	62
6.14.1	Equation-based nodes	62
6.14.2	Continuous evidence	64
6.14.3	Continuous inference	65
6.15	Hybrid models	67
6.16	Diagnosis	68
6.16.1	Diagnostic roles	69
6.16.2	Observation cost	71
6.16.3	Diagnostic session	72
6.16.4	Distance and entropy-based measures	75
6.17	Learning	76
6.17.1	Learning network structure	76
6.17.2	Learning network parameters	78
6.17.3	Validation	80
7.	Tutorials	83
7.1	Tutorial 1: Creating a Bayesian Network	84
7.1.1	Tutorial1.py	88

7.1.2	Tutorial1.java	89
7.1.3	Tutorial1.cs	91
7.1.4	Tutorial1.R	93
7.2	Tutorial 2: Inference with a Bayesian Network	95
7.2.1	Tutorial2.py	96
7.2.2	Tutorial2.java	97
7.2.3	Tutorial2.cs	98
7.2.4	Tutorial2.R	99
7.3	Tutorial 3: Exploring the contents of a model	101
7.3.1	Tutorial3.py	103
7.3.2	Tutorial3.java	104
7.3.3	Tutorial3.cs	106
7.3.4	Tutorial3.R	108
7.4	Tutorial 4: Creating the Influence Diagram	109
7.4.1	Tutorial4.py	111
7.4.2	Tutorial4.java	112
7.4.3	Tutorial4.cs	113
7.4.4	Tutorial4.R	114
7.5	Tutorial 5: Inference in an Influence Diagram	115
7.5.1	Tutorial5.py	116
7.5.2	Tutorial5.java	117
7.5.3	Tutorial5.cs	119
7.5.4	Tutorial5.R	120
7.6	Tutorial 6: Dynamic model	122
7.6.1	Tutorial6.py	125
7.6.2	Tutorial6.java	127
7.6.3	Tutorial6.cs	129
7.6.4	Tutorial6.R	131
7.7	Tutorial 7: Continuous model	132
7.7.1	Tutorial7.py	135
7.7.2	Tutorial7.java	136
7.7.3	Tutorial7.cs	138
7.7.4	Tutorial7.R	141
7.8	Tutorial 8: Hybrid model	142
7.8.1	Tutorial8.py	144
7.8.2	Tutorial8.java	145
7.8.3	Tutorial8.cs	147
7.8.4	Tutorial8.R	150
7.9	Tutorial 9: Diagnosis	152
7.9.1	Tutorial9.py	158

7.9.2	Tutorial9.java	160
7.9.3	Tutorial9.cs	162
7.9.4	Tutorial9.R	165
7.10	Tutorial 10: Structure learning	167
7.10.1	Tutorial10.py	172
7.10.2	Tutorial10.java	174
7.10.3	Tutorial10.cs	176
7.10.4	Tutorial10.R	179
8.	Equations reference	181
8.1	Operators	182
8.2	Random Number Generators	183
8.3	Statistical Functions	194
8.4	Arithmetic Functions	194
8.5	Combinatoric Functions	196
8.6	Trigonometric Functions	196
8.7	Hyperbolic Functions	197
8.8	Logical/Conditional functions	197
8.9	Custom Functions	198
9.	PySMILE reference	201
9.1	pysmile.AnnealedMapResults	202
9.2	pysmile.AnnealedMapTuning	202
9.3	pysmile.BayesianAlgorithmType	203
9.4	pysmile.CaseEvidenceInfo	203
9.5	pysmile.CaseEvidenceType	204
9.6	pysmile.DeMorganParentType	204
9.7	pysmile.DiagNetwork	204
9.8	pysmile.DiagResults	206
9.9	pysmile.DiscretizationInterval	206
9.10	pysmile.DocItemInfo	207
9.11	pysmile.EPISParams	207
9.12	pysmile.FaultInfo	207
9.13	pysmile.InfluenceDiagramAlgorithmType	208
9.14	pysmile.License	208
9.15	pysmile.MpeResults	208
9.16	pysmile.MultiFaultAlgorithmType	208

9.17	pysmile.Network	209
9.18	pysmile.NodeSensitivity	229
9.19	pysmile.NodeTemporalType	230
9.20	pysmile.NodeType	230
9.21	pysmile.ObservationInfo	231
9.22	pysmile.SMILEException	231
9.23	pysmile.SensitivityResults	231
9.24	pysmile.SingleFaultAlgorithmType	232
9.25	pysmile.TemporalInfo	232
9.26	pysmile.UnrollResults	232
9.27	pysmile.UserProperty	232
9.28	pysmile.ValueOfInfo	232
9.29	Learning	234
9.29.1	pysmile.learning.BayesianSearch	234
9.29.2	pysmile.learning.BkKnowledge	235
9.29.3	pysmile.learning.Curve	236
9.29.4	pysmile.learning.DataMatch	236
9.29.5	pysmile.learning.DataSet	236
9.29.6	pysmile.learning.EM	238
9.29.7	pysmile.learning.GreedyThickThinning	239
9.29.8	pysmile.learning.NaiveBayes	239
9.29.9	pysmile.learning.PC	239
9.29.10	pysmile.learning.Pattern	240
9.29.11	pysmile.learning.TAN	240
9.29.12	pysmile.learning.Validator	241

10. Acknowledgments 243

Index 0

This page is intentionally left blank.

Introduction

1 Introduction

Welcome to the SMILE Wrapper Programmer's Manual, Version 2.4.R1, built on 10/15/2025.

For the most recent version of this manual, please visit <https://support.bayesfusion.com/docs/>.

To download the software described in this manual, go to <http://download.bayesfusion.com>.

SMILE is a C++ software library for performing Bayesian inference and serves as the underlying API for **GeNIe**, BayesFusion's interactive modeling environment. To ensure that its functionality can be easily integrated into software written in other languages, BayesFusion, LLC provides wrapper libraries for Python, Java, R, and .NET. These products are:

- PySMILE (Python 2.7 and 3.x)
- jSMILE (Java and environments that can instantiate and use the JVM)
- rSMILE (R 4.x)
- SMILE.NET (.NET)

Each wrapper includes SMILE, so you do not need to download SMILE separately or be familiar with the C++ version in order to use the wrappers. We strive to maintain feature parity across wrappers—features available in one wrapper are generally available in the others.

BayesFusion has also developed SMILE.COM, a wrapper that exposes SMILE functionality through Windows COM (Common Object Model). The primary target audience for SMILE.COM is Microsoft Excel users, although the library can be used with any environment that supports COM. This manual does not include documentation for SMILE.COM.

If you are new to SMILE and would like to begin with an informal, tutorial-style introduction, please start with the Python, Java, R, or .NET section of [Platforms and Wrappers](#)^[16] (depending on the programming language you intend to use), followed by the [Hello SMILE Wrapper!](#)^[22] section. If you are an advanced user, please browse the *Table of Contents* or search for a specific topic.

This manual refers to several concepts assumed to be known to the reader, such as probability, utility, decision theory and decision analysis, Bayesian networks, and influence diagrams. For readers new to these concepts, the **GeNIe software and its manual** provide an interactive way to explore them. Since SMILE serves as GeNIe's API, practically every operation performed in GeNIe translates to calls to SMILE methods, making GeNIe familiarity extremely useful for learning SMILE. The GeNIe manual, along with all other BayesFusion documentation, is available at <https://support.bayesfusion.com/docs>.

Other resources, including introductions to probabilistic graphical models, are available at <https://www.bayesfusion.com/resources/>. Another useful resource is <https://repo.bayesfusion.com>, powered by **BayesBox**, our interactive model viewer and repository software. The repository contains more than 100 example Bayesian networks, hybrid Bayesian networks, dynamic Bayesian networks, and influence diagrams. BayesBox runs SMILE on the server side and calls into its API to calculate posterior probabilities after evidence is modified in the web browser.

Licensing

2 Licensing

SMILE wrappers are commercial products that require a development license. There are two types of development licenses: **Academic** and **Commercial**. An Academic license is free of charge for research and teaching use by individuals affiliated with an academic institution. All other uses require a Commercial license, which is available for purchase from BayesFusion, LLC.

Deployment of the SMILE library or SMILE wrappers—that is, embedding them into user programs—requires a deployment license. There are two types of deployment licenses:

- Server license — allows a program linked with SMILE to be deployed on a computer server.
- End-user program license — allows distribution of user programs that include SMILE.

For details on licensing and pricing, please contact BayesFusion, LLC.

The licensing system is implemented as a code fragment containing your license key, which must be executed as the first call to the SMILE wrapper used by your program. **This code is not included in the SMILE wrapper binary distribution**; it is personalized for you or your organization by BayesFusion, LLC. License keys are provided in a compressed .zip file containing Python, Java, R, C#, and VB.NET code. Six-month Academic licenses and free 30-day evaluation licenses can be obtained directly at <https://download.bayesfusion.com>.

What's new in SMILE 2

3 What's new in SMILE 2

Major new features in SMILE 2 are:

- Support for metalog distributions in continuous nodes. There is no new API for metalog distributions; equation nodes can use `Metalog` and `MetalogA` functions in their equations. For details, see the metalog entry in the [Random Number Generators](#)¹⁸³ section.
- Discrete nodes with outcomes based on numeric intervals or point values. See the [Discrete nodes and numeric domains](#)⁴⁸ section for details.

Version 2.2 introduces new diagnostic algorithms for single-fault and multi-fault [diagnosis](#)⁶⁸.

Platforms and wrappers

4 Platforms and wrappers

Feature parity between wrappers has been ensured, and similar names for the classes and methods have been used in their design. For example, the `Network` class is included in all wrappers. At the same time, the idioms of the specific target languages have been respected:

- `Network` class in PySMILE has had the `add_node` method implemented.
- `Network` class in jSMILE has had the `addNode` method implemented.
- `Network` class in rSMILE has had the `addNode` method implemented.
- `Network` class in SMILE.NET has had the `AddNode` method implemented.

As can be observed, the naming conventions of the different programming languages are reflected in the method names.

The remainder of this chapter describes the platform-specific details that are related to the use of SMILE Wrappers.

4.1 Python and PySMILE

To use PySMILE in your Python program, import the `pysmile` module:

```
import pysmile
```

Your license key can either be pasted directly into your Python code or imported as a module:

```
import pysmile_license
```

PySMILE binaries are available for Python 2.7 and Python 3.x. PySMILE consists of a single dynamically loaded library, whose name is platform-dependent:

- `pysmile.pyd` on Windows
- `pysmile.so` on Linux and macOS

You can manually download PySMILE binaries from BayesFusion's software download website or use `pip`, the Python package manager, to automate the download. The software download website provides binaries compatible with the most recent stable Python version, while the `pip` repository hosts builds for multiple Python releases.

To download using `pip`:

```
python -m pip install --no-cache-dir --index-url https://support.bayesfusion.com/pysmile-B/ pysmile
```

Replace **pysmile-B** with **pysmile-A** in the repository URL if you're using PySMILE academic. To force re-download of the `pysmile` module from the BayesFusion repository, add `--force-reinstall` to the command line.

Sonatype Nexus requires the `pip` repository URLs to end with `simple`. We maintain `pip` repository mirrors for users of Nexus at:

<https://support.bayesfusion.com/python/pysmile-A/simple/>
<https://support.bayesfusion.com/python/pysmile-B/simple/>

4.2 Java and jSMILE

jSMILE is compatible with JDK 8 and newer. See the [Maven](#)¹⁷ section below for information about using jSMILE in a Maven-based project, including the location of BayesFusion's Maven repository and POM integration with jSMILE's native library.

jSMILE consists of two parts: the JAR file and the native library, the latter being specific to your operating system. The JAR file, named `jsmile-x.y.z.jar` (where `x.y.z` is the version number), can be used like any other Java JAR.

The native library is used by the JAR code through the Java Native Interface (JNI). Its file name is platform-dependent:

- `jsmile.dll` on Windows
- `jsmile.so` on Linux
- `jsmile.jnilib` on macOS

In the static initializer of the `smile.Wrapper` class, jSMILE attempts to load its native library. If the `jsmile.native.library` system property is set, jSMILE calls `System.load` using the property value as the full path to the native library (relative paths are not allowed). Otherwise, `System.loadLibrary` is used. In this case, the native library must be located in one of the directories specified by the `java.library.path` property. Note that `java.library.path` cannot be modified once the JVM is running, but `jsmile.native.library` can be set dynamically with a call to `System.setProperty`.

Your license key should be pasted into the source code of your program.

The classes defined in jSMILE are located in `smile` and `smile.learning` packages. There are two types of classes:

- simple objects with public data members
- wrappers for C++ objects from the SMILE library

The second group of classes, which includes `smile.Network`, derives from `smile.Wrapper`. Each instance of these classes has an associated C++ object, which is deallocated either by the Java object's finalizer or by an explicit call to `Wrapper.Dispose`. To ensure timely deallocation of C++ resources, `Dispose` should be called as soon as the program no longer needs the object. If `Dispose` is not called, deallocation occurs during garbage collection in the finalizer. Java's `try-with-resources` statement can be used:

```
try (Network net = new Network()) {  
    // use the net object in this scope  
}
```

4.2.1 Maven

If your Java project uses Maven, you can integrate jSMILE using BayesFusion's Maven repository. The repository URL is <https://support.bayesfusion.com/maven-X>, where X is either A for jSMILE Academic or B for the commercial version of the library. The artifact ID is `jsmile` and the group ID is `com.bayesfusion`, regardless of the repository used.

In addition to the `jsmile` artifact, the repository provides three platform-specific `.zip` artifacts containing the native libraries:

- `jsmile-native-win64` for 64-bit Windows
- `jsmile-native-linux64` for 64-bit Linux
- `jsmile-native-macos` for macOS

Do not add references to these artifacts in the `<dependencies>` section of the POM file. Instead, use the `maven-dependency-plugin` to integrate them with your Maven build process and ensure that the native library is extracted from the artifact's `.zip` file into the project directory during the compile phase. For projects that only run on one of the three supported platforms, the artifact ID specified in the plugin configuration can directly reference the corresponding artifact listed above. Alternatively, you can use Maven's OS-based profile activation to select the appropriate native library. The POM file below demonstrates this approach. Note that the CPU architecture is not checked (a 64-bit environment is assumed for both Windows and Linux). In this example, the extracted library is placed in `${project.basedir}`, and the referenced `jSMILE` version is set to `2.2.0`.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.bayesfusion</groupId>
  <artifactId>test-maven-repo</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <repositories>
    <repository>
      <id>bayesfusion-repo</id>
      <name>BayesFusion Repository</name>
      <url>https://support.bayesfusion.com/maven-B</url>
    </repository>
  </repositories>

  <dependencies>
    <dependency>
      <groupId>com.bayesfusion</groupId>
      <artifactId>jsmile</artifactId>
      <version>2.2.0</version>
    </dependency>
  </dependencies>

  <profiles>
    <profile>
      <id>MacNative</id>
      <activation>
        <os>
          <family>mac</family>
        </os>
      </activation>
      <properties>
        <nativeSuffix>macos</nativeSuffix>
      </properties>
    </profile>
    <profile>
      <id>WinNative</id>
      <activation>
        <os>
```

```

        <family>Windows</family>
      </os>
    </activation>
  </properties>
  <nativeSuffix>win64</nativeSuffix>
</properties>
</profile>
<profile>
  <id>LinuxNative</id>
  <activation>
    <os>
      <family>Unix</family>
    </os>
  </activation>
  <properties>
    <nativeSuffix>linux64</nativeSuffix>
  </properties>
</profile>
</profiles>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-dependency-plugin</artifactId>
      <executions>
        <execution>
          <id>resource-dependencies</id>
          <phase>compile</phase>
          <goals>
            <goal>unpack</goal>
          </goals>
          <configuration>
            <artifactItems>
              <artifactItem>
                <groupId>com.bayesfusion</groupId>
                <artifactId>jsmile-native-${nativeSuffix}</artifactId>
                <version>2.2.0</version>
                <type>zip</type>
                <overWrite>true</overWrite>
                <includes>*jsmile*</includes>
              </artifactItem>
            </artifactItems>
            <outputDirectory>${project.basedir}</outputDirectory>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

</project>

```

4.3 .NET and SMILE.NET

SMILE.NET, which is compatible with .NET framework 4.x, consists of a single mixed-mode assembly named `smilenet.dll`. To use the library, simply add `smilenet.dll` as a reference in your project.

We build SMILE.NET using C++/CLI compiler in Visual Studio 2013 to create a seamless package containing both native code (the C++ SMILE library) and .NET wrapper code. This compiler enforces the use of the dynamic C++ runtime library. This means that `smilenet.dll` has a runtime dependency on VS2013 C++ runtime. Your Windows system is likely to have this component already installed. However, if you are getting error messages referring to missing DLLs like `MSVCP120.DLL` or `MSVCR120.DLL`, you need to download and install the "Visual C++ Redistributable Package for Visual Studio 2013" from Microsoft's website.

Your license key must be included as the first call to the SMILE.NET wrapper in your program. Keys are provided in both C# and VB.NET code and should be pasted into your application's source.

The classes defined in SMILE.NET are located in `Smile` and `Smile.Learning` namespaces. There are two types of classes:

- simple objects with public data members
- wrappers for C++ objects from SMILE library

The second group of classes, which includes `Smile.Network`, is derived from `Smile.WrappedObject` and also implements .NET's `IDisposable` interface. Each object instance of these classes has a related C++ object, which is deallocated by the .NET object's finalizer or its `IDisposable.Dispose` method. You can use C#'s `using` statement to ensure deterministic finalization, otherwise the deallocation will be performed in the finalizer during garbage collection.

```
using (Network net = new Network())
{
    // use the net object in this scope
}
```

4.4 R and rSMILE

rSMILE is distributed as a compiled R package. rSMILE binaries are available for R 4.x.

To install rSMILE, use `install.packages`:

```
install.packages(<path-to-local-copy-of-rSMILE-package>)
```

Once the package is installed, use the library function to enable rSMILE functionality:

```
library(rSMILE)
```

Your license key can be pasted directly into your R code, or read from the license file:

```
source("License.R")
```

From December 2019 onwards, the licensing keys for R are generated in the format compatible with rSMILE and are ready to be used with rSMILE. If you obtained your SMILE licensing key before December 2019, your `License.R` file uses the syntax for jSMILE and rJava. To modify your licensing key, replace

```
.jnew("smile.License",paste(
with
```

```
rSMILE::License(paste(
```

Another required modification applies to the numeric part of the licensing key: replace `.jbyte` with `as.integer`. Please contact BayesFusion if you encounter any problems with the modification, or, if you're an academic user, generate new licensing key at our academic downloads site.

Hello, SMILE Wrapper!

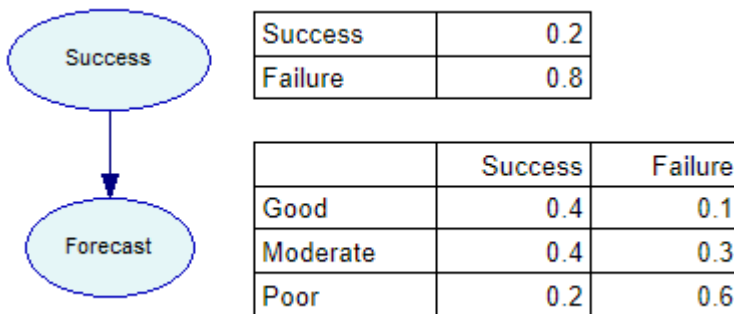
5 Hello, SMILE Wrapper!

In this section, we will show how SMILE can load and use a model created in GeNIe to perform useful work. We will use the model developed in the GeNIe on-line help (Section *Hello GeNIe!*). The model for this problem is available as one of the example networks (file `VentureBN.xdsl`). If you have GeNIe installed, you can copy the file into your working directory. The same file is also included in the zip file containing all source code for tutorials, available from <http://support.bayesfusion.com/docs>.

Alternatively, create a file named `VentureBN.xdsl` with any text editor by copying the content of the [VentureBN.xdsl](#)^[22] section below.

5.1 Success/Forecast model

The model encodes information pertaining to a problem faced by a venture capitalist, who considers a risky investment in a startup company. A major source of uncertainty about her investment is the success of the company. She is aware of the fact that only around 20% of all start-up companies succeed. She can reduce this uncertainty somewhat by asking expert opinion. Her expert, however, is not perfect in his forecasts. Of all start-up companies that eventually succeed, he judges about 40% to be good prospects, 40% to be moderate prospects, and 20% to be poor prospects. Of all start-up companies that eventually fail, he judges about 10% to be good prospects, 30% to be moderate prospects, and 60% to be poor prospects.



5.2 VentureBN.xdsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<smile version="1.0" id="VentureBN" numsamples="1000">
  <nodes>
    <cpt id="Success">
      <state id="Success" />
      <state id="Failure" />
      <probabilities>0.2 0.8</probabilities>
    </cpt>
    <cpt id="Forecast">
      <state id="Good" />
      <state id="Moderate" />
      <state id="Poor" />
      <parents>Success</parents>
      <probabilities>
        0.4 0.4 0.2 0.1 0.3 0.6
      </probabilities>
    </cpt>
  </nodes>
```

```

<extensions>
  <genie version="1.0" app="GeNIe 2.1.1104.2"
    name="VentureBN"
    faultnameformat="nodestate">
    <node id="Success">
      <name>Success of the venture</name>
      <interior color="e5f6f7" />
      <outline color="0000bb" />
      <font color="000000" name="Arial" size="8" />
      <position>54 11 138 62</position>
    </node>
    <node id="Forecast">
      <name>Expert forecast</name>
      <interior color="e5f6f7" />
      <outline color="0000bb" />
      <font color="000000" name="Arial" size="8" />
      <position>63 105 130 155</position>
    </node>
  </genie>
</extensions>
</smile>

```

5.3 The program

We will show how to load this model using SMILE, how to enter observations (evidence), how to perform inference, and how to retrieve the results of SMILE's calculations. Three complete source code versions (Java, Python, R, C#) are included below. Note that you'll need to use your SMILE license key. See the [Licensing](#)¹² section of this manual if you want to obtain your academic or trial license key. See [Platforms and Wrappers](#)¹⁶ section for language-specific information on licensing.

Our network object - an instance of `Network` class - is declared as local variable. We proceed to read the XDSL file from disk:

Python

```
net.read_file("VentureBN.xdsl")
```

Java

```
net.readFile("VentureBN.xdsl");
```

C#

```
net.ReadFile("VentureBN.xdsl");
```

R

```
net$readFile("VentureBN.xdsl")
```

We want to set the evidence on the *Forecast* node to *Moderate*. The wrappers can use node and outcome identifiers directly:

Python

```
net.set_evidence("Forecast", "Moderate")
```

Java

```
net.setEvidence("Forecast", "Moderate");
```

C#

```
net.SetEvidence("Forecast", "Moderate");
```

R

```
net$setEvidence("Forecast", "Moderate")
```

Now we can update the network:

Python

```
net.update_beliefs()
```

Java

```
net.updateBeliefs();
```

C#

```
net.UpdateBeliefs();
```

R

```
net$updateBeliefs()
```

After network update we can retrieve the posterior probabilities of the *Success* node:

Python

```
beliefs = net.get_node_value("Success")
```

Java

```
double[] beliefs = net.getNodeValue("Success");
```

C#

```
double[] beliefs = net.GetNodeValue("Success");
```

R

```
beliefs <- net$getNodeValue("Success")
```

To print the probabilities, we simply iterate over the elements of the returned numeric array. Each probability printed to the console is preceded by the identifier of the node outcome. If you compile and run the program, the output should be:

```
Success=0.25
Failure=0.75
```

“Success” and “Failure” are outcome identifiers of the *Success* node, which in Python are returned by the `Network.get_outcome_id` method. At this point you should easily derive the equivalent method names for Java, C# and R from Python method name.

Note that in case of any of method calls failing, the exception would be thrown. SMILE wrappers do not use error codes on failure.

We will build upon the simple network described in this chapter in the [Tutorials](#)^[84] section of this manual.

5.3.1 Hello.py

```
import pysmile

# pysmile_license is your license key
import pysmile_license

def hello_smile():
    net = pysmile.Network()
    net.read_file("VentureBN.xdsl");

    net.set_evidence("Forecast", "Moderate")
    net.update_beliefs()

    beliefs = net.get_node_value("Success")
    for i in range(0, len(beliefs)):
        print(net.get_outcome_id("Success", i) + "=" + str(beliefs[i]))

hello_smile()
```

5.3.2 Hello.java

```
import smile.*;

public class Hello {
    public static void main(String[] args) {
        System.out.println("Insert your licensing key below.");
        // new smile.License(...);

        Network net = new Network();
        net.readFile("VentureBN.xdsl");
        net.setEvidence("Forecast", "Moderate");
        net.updateBeliefs();
        double[] beliefs = net.getNodeValue("Success");
        for (int i = 0; i < beliefs.length; i++) {
            System.out.println(net.getOutcomeId("Success", i) + " = " + beliefs[i]);
        }
    }
}
```

5.3.3 Hello.cs

```
using System;
using Smile;

namespace SmileNetTutorial
{
    class Hello
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Insert your SMILE licensing key here.");
            // new Smile.License(...);

            Network net = new Network();
            net.ReadFile("VentureBN.xdsl");
            net.SetEvidence("Forecast", "Moderate");
            net.UpdateBeliefs();
            double[] beliefs = net.GetNodeValue("Success");
            for (int i = 0; i < beliefs.Length; i++)
            {
                Console.WriteLine("{0} = {1}", net.GetOutcomeId("Success", i), beliefs[i]);
            }
        }
    }
}
```

5.3.4 Hello.R

```
library(rSMILE)

# License.R is your licensing key
source("License.R")

net <- Network()
net$readFile("VentureBN.xdsl")
net$setEvidence("Forecast", "Moderate")
net$updateBeliefs()
```

```
beliefs <- net$getNodeValue("Success")
for (i in 1:length(beliefs)) {
  cat(sprintf("%s = %f\n", net$getOutcomeId("Success", i-1L), beliefs[i]))
}
```

Using SMILE Wrappers

6 Using SMILE Wrappers

6.1 Error handling

Python, Java and .NET SMILE Wrappers throw exceptions when the underlying C++ SMILE calls fail. The exception types are:

Python

`pysmile.SMILEException`

Java

`smile.SMILEException`

C#

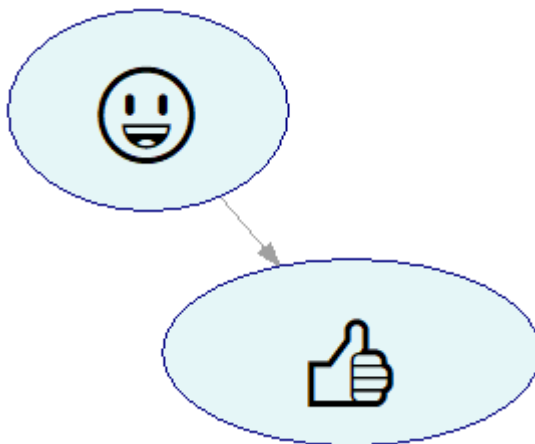
`Smile.SmileException`

R's error handling is a form of exception handling, implemented via its condition system. The rSMILE package raises an error condition if underlying C++ SMILE calls fail. This error condition can be handled in the same way as other errors in the R environment, namely through the `try()` and `tryCatch()` functions.

6.2 Identifiers

SMILE requires nodes in the network to have unique textual identifiers (in addition to potentially non-unique names and descriptions). A valid identifier starts with a letter and contains letters, decimal digits, and underscores. Rules for identifier syntax are also used in other parts of the SMILE API. For example, the names of the [custom functions](#)¹⁹⁸ and [user properties](#)⁴⁵ must be valid identifiers.

The letter in the identifier is defined as characters from a-z and A-Z range, but also any Unicode character with the codepoint greater than 127 (the Unicode character category is not checked). This allows for identifiers containing mathematical symbols, letters from alphabets other than Latin, or even emojis. Of course the program that renders the graphical view of your network must be able to use Unicode. Here is how a simple two node network with emoji node ids may look in GeNIe or BayesBox:



6.3 Networks, nodes and arcs

The most important class defined by the SMILE Wrappers is `Network`. Objects of this class act as containers for nodes and are responsible for their creation and destruction. Nodes and arcs are always created and removed by invoking methods of the `Network` class, and access to them is always performed through the `Network` object that

contains them. There are no separate classes representing nodes or arcs; the `Network` object serves as a facade for the complex underlying data structure.

6.3.1 Network

To create a network object, use a default constructor. For platform-specific information about the lifetime of `Network` instances, see the [Platforms and Wrappers](#)^[16] section.

Python

```
import pysmile
...
net = pysmile.Network()
```

Java

```
import smile.*;
...
Network net = new Network();
```

C#

```
using Smile;
...
Network net = new Network();
```

R

```
library(rSMILE)
...
net <- Network()
```

6.3.2 Nodes

Within a network, each node is uniquely identified by its **handle**, a non-negative integer that is preserved when the network is copied. Node handles are not guaranteed to be consecutive or start from any particular value.

In addition to handles, each node has a unique textual identifier within its containing network. This identifier can be specified when creating the node via the `Network.add_node` method and may be changed later. A node's identifier can be converted to its handle using `Network.get_node`, and conversely, a handle can be converted to an identifier using `Network.get_node_id`. See the [Identifiers](#)^[28] section for rules on valid node identifiers.

All methods of the `Network` class that operate on nodes accept either handles or identifiers. Note that methods using identifiers perform an **O(N)** string lookup (where N is the number of nodes), whereas methods using handles require only an **O(1)** validity check.

6.3.2.1 Node types

When creating a node, you must specify its **type**. The type is an integer enumerated in the wrapper-specific way. The available node types are:

- CPT: Discrete chance node with conditional probability table
- TRUTHTABLE: Discrete deterministic node
- NOISY_MAX: Discrete chance node with Noisy-MAX canonical parameters
- NOISY_ADDER: Discrete chance node with Noisy-Adder canonical parameters
- DEMORGAN: Binary DeMorgan (qualitative) node

- **DECISION**: Discrete decision node
- **UTILITY**: Utility node
- **MAU**: Multi-attribute utility node
- **EQUATION**: Continuous node defined by equation, either chance or deterministic

6.3.2.2 Creating and deleting nodes

Use the following methods of the `Network` class to add or delete nodes. The integer value returned by the node-adding method is the **handle** of the newly created node, which can be used in subsequent SMILE calls.

Python

```
add_node(node_type: pysmile.NodeType, node_id: str) -> int
add_node(node_type: pysmile.NodeType) -> int
delete_node(node: int | str) -> None
```

Java

```
int addNode(int nodeType, String nodeId)
int addNode(int nodeType)
void deleteNode(int nodeHandle)
void deleteNode(String nodeId)
```

C#

```
int AddNode(int nodeType, string nodeId)
int AddNode(int nodeType)
void DeleteNode(int nodeHandle)
void DeleteNode(string nodeId)
```

R

```
nodeHandle <- addNode(nodeType, nodeId)
deleteNode(nodeIdOrHandle)
```

When creating a node, different SMILE wrappers expose node type identifiers in different ways: in Java and C#, they are nested classes with static integer values; in R, they are members of a network instance; and in Python, they are provided as a separate enum type.

Python

```
net.add_node(pysmile.NodeType.CPT, "node1")
```

Java

```
net.addNode(Network.NodeType.CPT, "node1");
```

C#

```
net.AddNode(Network.NodeType.CPT, "node1");
```

R

```
net$addNode(net$NodeType$CPT, "node1")
```

6.3.2.3 Iterating over nodes

To iterate over nodes in the network, use `Network.get_first_node` and `Network.get_next_node`. Note the loop exit condition: iteration continues only while the returned handle is greater than or equal to zero.

Python

```
h = net.get_first_node()
while (h >= 0):
    # do something with net and node identified by h
    h = net.get_next_node(h)
```

Java

```
for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h)) {
    // do something with net and node identified by h
}
```

C#

```
for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
{
    // do something with net and node identified by h
}
// alternatively, iterate directly via IEnumerable implemented by Network
foreach (int h in net)
{
    // do something with net and node identified by h
}
```

R

```
h <- net$getFirstNode()
while (h >= 0) {
    # do something with net and node identified by h
    h <- net$getNextNode(h)
}
```

The following methods allow you to obtain an array of all node handles, an array of all node identifiers, or the number of nodes in the network:

Python

```
get_all_nodes() -> List[int]
get_all_node_ids() -> List[str]
get_node_count() -> int
```

Java

```
int[] getAllNodes()
String[] getAllNodeIds()
int getNodeCount()
```

C#

```
int[] GetAllNodes()
string[] GetAllNodeIds()
int NodeCount { get; }
```

R

```
allNodeHandles <- getAllNodes()
allNodeIds <- getAllNodeIds()
nodeCount <- getNodeCount()
```

6.3.2.4 Node definition

The node definition specifies how a node interacts with other nodes in the network. It is stored as part of the network when the network is saved to a file or serialized. Equation nodes are defined by their equation (see the [Equation-based nodes](#)⁶² section for details), while other node types, such as discrete chance nodes and utility nodes, have a definition that includes numeric data. Use the following methods to read or write the numeric part of a node definition:

Python

```
set_node_definition(node: int | str, node_definition: List[float]) -> None
get_node_definition(node: int | str) -> List[float]
```

Java

```
void setNodeDefinition(int nodeHandle, double[] definition);
void setNodeDefinition(String nodeId, double[] definition);
double[] getNodeDefinition(int nodeHandle);
double[] getNodeDefinition(String nodeId);
```

C#

```
void SetNodeDefinition(int nodeHandle, double[] definition);
void SetNodeDefinition(string nodeId, double[] definition);
double[] GetNodeDefinition(int nodeHandle);
double[] GetNodeDefinition(string nodeId);
```

R

```
setNodeDefinition(nodeIdOrHandle)
nodeDef <- getNodeDefinition(nodeIdOrHandle)
```

The numeric data stored in the node definition is internally represented as a flattened multidimensional array, where each dimension corresponds to one discrete parent node plus the node itself. For more details on the storage and indexing of these arrays, see the [Multidimensional arrays](#)^[39] section.

For discrete nodes (including general chance nodes), the definition also includes a list of outcomes representing the possible states of the node. Outcomes can be defined in one of four ways:

1. Identifiers only – each outcome has a unique string label within the node. Note that different nodes may use the same outcome identifiers.
2. Identifiers with numeric intervals – each outcome has a string label and an associated numeric interval.
3. Numeric intervals only – outcomes are defined purely by numeric intervals without identifiers.
4. Identifiers with numeric point values – each outcome has a string label and a numeric point value (point values cannot exist without identifiers).

For more details on discrete nodes and the use of numeric domains, see the [Discrete nodes and numeric domains](#)^[48] section.

Outcomes can be added, deleted, or renamed regardless of how they are defined.

Python

```
get_outcome_count(node: int | str) -> int
add_outcome(node: int | str, outcome_id: str) -> None
insert_outcome(node: int | str, position: int, outcome_id: str) -> None
delete_outcome(node: int | str, outcome: int | str) -> None
set_outcome_id(node: int | str, outcome_index: int, id: str) -> None
get_outcome_id(node: int | str, outcome_index: int) -> str
get_outcome_ids(node: int | str) -> List[str]
```

Java

```
int getOutcomeCount(int nodeHandle);
int getOutcomeCount(String nodeId);
void addOutcome(int nodeHandle, String outcomeId);
void addOutcome(String nodeId, String outcomeId);
void insertOutcome(int nodeHandle, int position, String outcomeId);
void insertOutcome(String nodeId, int position, String outcomeId);
void deleteOutcome(int nodeHandle, int outcomeIndex);
void deleteOutcome(String nodeId, int outcomeIndex);
void deleteOutcome(int nodeHandle, String outcomeId);
void deleteOutcome(String nodeId, String outcomeId);
void setOutcomeId(int nodeHandle, int outcomeIndex, String id);
void setOutcomeId(String nodeId, int outcomeIndex, String id);
String getOutcomeId(int nodeHandle, int outcomeIndex);
String getOutcomeId(String nodeId, int outcomeIndex);
String[] getOutcomeIds(int nodeHandle);
String[] getOutcomeIds(String nodeId);
```

C#

```

int GetOutcomeCount(int nodeHandle);
int GetOutcomeCount(string nodeId);
void AddOutcome(int nodeHandle, string outcomeId);
void AddOutcome(string nodeId, string outcomeId);
void InsertOutcome(int nodeHandle, int position, string outcomeId);
void InsertOutcome(string nodeId, int position, string outcomeId);
void DeleteOutcome(int nodeHandle, int outcomeIndex);
void DeleteOutcome(string nodeId, int outcomeIndex);
void DeleteOutcome(int nodeHandle, string outcomeId);
void DeleteOutcome(string nodeId, string outcomeId);
void SetOutcomeId(int nodeHandle, int outcomeIndex, string id);
void SetOutcomeId(string nodeId, int outcomeIndex, string id);
string GetOutcomeId(int nodeHandle, int outcomeIndex);
string GetOutcomeId(string nodeId, int outcomeIndex);
string[] GetOutcomeIds(int nodeHandle);
string[] GetOutcomeIds(string nodeId);

```

R

```

outcomeCount <- getOutcomeCount(nodeIdOrHandle)
addOutcome(nodeIdOrHandle, outcomeId)
insertOutcome(nodeIdOrHandle, position, outcomeId)
deleteOutcome(nodeIdOrHandle, outcomeIndexOrId)
setOutcomeId(nodeIdOrHandle, outcomeIndex, id)
outcomeId <- getOutcomeId(nodeIdOrHandle, outcomeIndex)
outcomeIds <- getOutcomeIds(nodeIdOrHandle)

```

6.3.2.5 Node value

The value of a node contains the results calculated by the inference algorithm, typically the marginal probability distribution or expected utilities. Unlike the node definition, the node value is **transient** and is **not** stored as part of the network during I/O operations. Each node value also includes a **validity flag**, which is set to true by the inference algorithm once the calculations for the node have been successfully completed. Value-reading methods should only be called when this flag indicates that the value is valid.

In influence diagrams, the node value is computed for multiple sets of decisions. To correctly interpret these results, you need to retrieve the indexing parents of the node value. These indexing parents may or may not coincide with the node's structural parents. For more details, refer to [Influence diagrams](#)^[54] section. When indexing parents are present, the values are organized as flattened vectors representing multidimensional arrays. For details on interpreting these arrays and converting between linear and multidimensional forms, see the [Multidimensional arrays](#)^[39] section.

Additional considerations apply in two cases:

- For dynamic Bayesian networks, the handling of node values is explained in the [Temporal beliefs](#)^[62] section.
- For equation nodes, the interpretation of the value differs from standard chance or utility nodes; see the [Equation-based nodes](#)^[62] section for details.

The following methods allow you to check the validity of a node value, retrieve the value itself, and obtain information about its indexing parents (if present):

Python

```

is_value_valid(node: int | str) -> bool
get_node_value(node: int | str) -> List[float]
get_value_indexing_parents(node: int | str) -> List[int]
get_value_indexing_parent_ids(node: int | str) -> List[str]

```

Java

```

boolean isValueValid(int nodeHandle);
boolean isValueValid(String nodeId);
double[] getNodeValue(int nodeHandle);
double[] getNodeValue(String nodeId);
int[] getValueIndexingParents(int nodeHandle);
int[] getValueIndexingParents(String nodeId);
String[] getValueIndexingParentIds(int nodeHandle);
String[] getValueIndexingParentIds(String nodeId);

```

C#

```

boolean IsValueValid(int nodeHandle);
boolean IsValueValid(String nodeId);
double[] GetNodeValue(int nodeHandle);
double[] GetNodeValue(String nodeId);
int[] GetValueIndexingParents(int nodeHandle);
int[] GetValueIndexingParents(string nodeId);
string[] GetValueIndexingParentIds(int nodeHandle);
string[] GetValueIndexingParentIds(string nodeId);

```

R

```

valid <- isValueValid(nodeIdOrHandle)
nodeValue <- getNodeValue(nodeIdOrHandle)
indexingParents <- get_value_indexing_parents(nodeIdOrHandle)
indexingParentIds <- get_value_indexing_parent_ids(nodeIdOrHandle)

```

6.3.2.6 Node evidence

The output from inference algorithms depends on node definitions and the **evidence** set on nodes. For discrete nodes, evidence is specified by outcome index or identifier; for continuous nodes, it is a numeric value, with related methods including “cont” in their names (e.g., `Network.get_cont_evidence`). In [dynamic Bayesian networks](#)⁵⁴ (DBNs), evidence is set for specific time slices, with corresponding methods including “temporal” (e.g., `Network.get_temporal_evidence`).

Nodes can also have **propagated evidence**, implied by other evidence in the network, and **virtual evidence**, a probability distribution over discrete node outcomes.

Evidence is **transient**: it is not stored when the network is saved or serialized, and must be reapplied when a network is reloaded. For users who need to store multiple evidence sets persistently, SMILE provides a [case manager](#)⁴⁵ functionality that allows different evidence cases to be saved directly in the network object and serialized along with it.

The methods related to evidence include:

Python

```

is_evidence(node: int | str) -> bool
is_propagated_evidence(node: int | str) -> bool
is_real_evidence(node: int | str) -> bool
is_virtual_evidence(node: int | str) -> bool
is_temporal_evidence(node: int | str) -> bool
has_temporal_evidence(node: int | str) -> bool
clear_evidence(node: int | str) -> None
clear_temporal_evidence(node: int | str, time_slice: int) -> None
get_evidence(node: int | str) -> int
get_evidence_id(node: int | str) -> str
get_cont_evidence(node: int | str) -> float
get_virtual_evidence(node: int | str) -> List[float]
get_temporal_evidence(node: int | str, time_slice: int) -> int
get_temporal_evidence_id(node: int | str, time_slice: int) -> str
get_temporal_virtual_evidence(node: int | str, time_slice: int) -> List[float]

```

```

set_evidence(node: int | str, outcome: int | str) -> None
set_cont_evidence(node: int | str, value: float) -> None
set_virtual_evidence(node: int | str, distribution: List[float]) -> None
set_temporal_evidence(node: int | str, time_slice: int, outcome: int | str) -> None
set_temporal_virtual_evidence(node: int | str, time_slice: int, distribution: List[float]) -> None

```

Java

```

boolean isEvidence(int nodeHandle);
boolean isEvidence(String nodeId);
boolean isPropagatedEvidence(int nodeHandle);
boolean isPropagatedEvidence(String nodeId);
boolean isRealEvidence(int nodeHandle);
boolean isRealEvidence(String nodeId);
boolean isVirtualEvidence(int nodeHandle);
boolean isVirtualEvidence(String nodeId);
boolean isTemporalEvidence(int nodeHandle);
boolean isTemporalEvidence(String nodeId);
boolean hasTemporalEvidence(int nodeHandle);
boolean hasTemporalEvidence(String nodeId);
void clearEvidence(int nodeHandle);
void clearEvidence(String nodeId);
int getEvidence(int nodeHandle);
String getEvidenceId(String nodeId);
void setEvidence(int nodeHandle, int outcomeIndex);
void setEvidence(String nodeId, String outcomeId);
double getContEvidence(int nodeHandle);
double getContEvidence(String nodeId);
void setContEvidence(int nodeHandle, double value);
void setContEvidence(String nodeId, double value);
double[] getVirtualEvidence(int nodeHandle);
double[] getVirtualEvidence(String nodeId);
void setVirtualEvidence(int nodeHandle, double[] distribution);
void setVirtualEvidence(String nodeId, double[] distribution);
void clearTemporalEvidence(int nodeHandle, int timeSlice);
void clearTemporalEvidence(String nodeId, int timeSlice);
int getTemporalEvidence(int nodeHandle, int timeSlice);
String getTemporalEvidenceId(String nodeId, int timeSlice);
double[] getTemporalVirtualEvidence(int nodeHandle, int timeSlice);
void setTemporalEvidence(int nodeHandle, int timeSlice, int outcomeIndex);
void setTemporalEvidence(String nodeId, int timeSlice, String outcomeId);
void setTemporalVirtualEvidence(int nodeHandle, int timeSlice, double[] distribution);
void setTemporalVirtualEvidence(String nodeId, int timeSlice, double[] distribution);

```

C#

```

bool IsEvidence(int nodeHandle);
bool IsEvidence(string nodeId);
bool IsPropagatedEvidence(int nodeHandle);
bool IsPropagatedEvidence(string nodeId);
bool IsRealEvidence(int nodeHandle);
bool IsRealEvidence(string nodeId);
bool IsVirtualEvidence(int nodeHandle);
bool IsVirtualEvidence(string nodeId);
bool IsTemporalEvidence(int nodeHandle);
bool IsTemporalEvidence(string nodeId);
bool HasTemporalEvidence(int nodeHandle);
bool HasTemporalEvidence(string nodeId);
void ClearEvidence(int nodeHandle);
void ClearEvidence(string nodeId);
void ClearTemporalEvidence(int nodeHandle, int timeSlice);
void ClearTemporalEvidence(string nodeId, int timeSlice);
int GetEvidence(int nodeHandle);
int GetEvidence(string nodeId);

```

```

string GetEvidenceId(int nodeHandle);
string GetEvidenceId(string nodeId);
double GetContEvidence(int nodeHandle);
double GetContEvidence(string nodeId);
double[] GetVirtualEvidence(int nodeHandle);
double[] GetVirtualEvidence(string nodeId);
int GetTemporalEvidence(int nodeHandle, int timeSlice);
int GetTemporalEvidence(string nodeId, int timeSlice);
string GetTemporalEvidenceId(int nodeHandle, int timeSlice);
string GetTemporalEvidenceId(string nodeId, int timeSlice);
double[] GetTemporalVirtualEvidence(int nodeHandle, int timeSlice);
double[] GetTemporalVirtualEvidence(string nodeId, int timeSlice);
void SetEvidence(int nodeHandle, int outcome);
void SetEvidence(string nodeId, int outcome);
void SetEvidence(int nodeHandle, string outcomeId);
void SetEvidence(string nodeId, string outcomeId);
void SetContEvidence(int nodeHandle, double value);
void SetContEvidence(string nodeId, double value);
void SetVirtualEvidence(int nodeHandle, double[] distribution);
void SetVirtualEvidence(string nodeId, double[] distribution);
void SetTemporalEvidence(int nodeHandle, int timeSlice, int outcome);
void SetTemporalEvidence(string nodeId, int timeSlice, int outcome);
void SetTemporalEvidence(int nodeHandle, int timeSlice, string outcomeId);
void SetTemporalEvidence(string nodeId, int timeSlice, string outcomeId);
void SetTemporalVirtualEvidence(int nodeHandle, int timeSlice, double[] distribution);
void SetTemporalVirtualEvidence(string nodeId, int timeSlice, double[] distribution);

```

R

```

evidenceStatus <- isEvidence(nodeIdOrHandle)
propagatedEvidenceStatus <- isPropagatedEvidence(nodeIdOrHandle)
realEvidenceStatus <- isRealEvidence(nodeIdOrHandle)
virtualEvidenceStatus <- isVirtualEvidence(nodeIdOrHandle)
temporalEvidenceStatus <- isTemporalEvidence(nodeIdOrHandle)
hasTemporal <- hasTemporalEvidence(nodeIdOrHandle)
clearEvidence(nodeIdOrHandle)
clearTemporalEvidence(nodeIdOrHandle, timeSlice)
evidenceValue <- getEvidence(nodeIdOrHandle)
evidenceId <- getEvidenceId(nodeIdOrHandle)
contEvidenceValue <- getContEvidence(nodeIdOrHandle)
virtualEvidenceValues <- getVirtualEvidence(nodeIdOrHandle)
temporalEvidenceValues <- getTemporalEvidence(nodeIdOrHandle, timeSlice)
temporalEvidenceIds <- getTemporalEvidenceId(nodeIdOrHandle, timeSlice)
temporalVirtualValues <- getTemporalVirtualEvidence(nodeIdOrHandle, timeSlice)
setEvidence(nodeIdOrHandle, outcomeOrId)
setContEvidence(nodeIdOrHandle, value)
setVirtualEvidence(nodeIdOrHandle, distribution)
setTemporalEvidence(nodeIdOrHandle, timeSlice, outcomeOrId)
setTemporalVirtualEvidence(nodeIdOrHandle, timeSlice, distribution)

```

6.3.2.7 Other node attributes

Node attributes control how a node appears in the graphical interface of GeNIe, BayesBox or other SMILE-based tools. These attributes **do not** affect probabilistic calculations or inference results; they are purely visual. Each node has textual attributes (name and optional description), positional attributes (coordinates and size), and color attributes (background, text, and border colors).

Nodes can also be assigned to **submodels** to help organize complex networks. For more details, see the [Submodels](#) ⁴⁵ chapter.

Textual attributes:

Python

```

set_node_name(node: int | str, name: str) -> None
get_node_name(node: int | str) -> str
set_node_description(node: int | str, description: str) -> None
get_node_description(node: int | str) -> str

```

Java

```

void setNodeName(int nodeHandle, String name);
void setNodeName(String nodeId, String name);
String getNodeName(int nodeHandle);
String getNodeName(String nodeId);
void setNodeDescription(int nodeHandle, String description);
void setNodeDescription(String nodeId, String description);
String getNodeDescription(int nodeHandle);

```

C#

```

void SetNodeName(int nodeHandle, string name);
void SetNodeName(string nodeId, string name);
string GetNodeName(int nodeHandle);
string GetNodeName(string nodeId);
void SetNodeDescription(int nodeHandle, string description);
void SetNodeDescription(string nodeId, string description);
string GetNodeDescription(int nodeHandle);

```

R

```

setNodeName(nodeIdOrHandle, name)
nodeName <- getNodeName(nodeIdOrHandle)
setNodeDescription(nodeIdOrHandle, description)
nodeDescription <- getNodeDescription(nodeIdOrHandle)

```

Positional attributes:

Python

```

set_node_position(node: int | str, rect: List[int]) -> None
get_node_position(node: int | str) -> List[int]

```

Java

```

void setNodePosition(int nodeHandle, int x, int y, int width, int height);
void setNodePosition(String nodeId, int x, int y, int width, int height);
void setNodePosition(int nodeHandle, java.awt.Rectangle rect)
void setNodePosition(String nodeId, java.awt.Rectangle rect)
java.awt.Rectangle getNodePosition(int nodeHandle);
java.awt.Rectangle getNodePosition(String nodeId);

```

C#

```

void SetNodePosition(int nodeHandle, System.Drawing.Rectangle rect);
void SetNodePosition(string nodeId, System.Drawing.Rectangle rect);
System.Drawing.Rectangle GetNodePosition(int nodeHandle);
System.Drawing.Rectangle GetNodePosition(string nodeId);

```

R

```

set_node_position(nodeIdOrHandle, rect)
position <- get_node_position(nodeIdOrHandle)

```

Node colors in SMILE are stored in the Windows API COLORREF format, a legacy of SMILE's early development around 2000 when Windows was the dominant platform. In this representation, the red, green, and blue components each occupy one byte (8 bits) and are ordered as BB GG RR in memory (least significant byte is red, most significant byte is blue). This format is preserved for compatibility, and color values passed to or returned from SMILE methods must follow this convention.

For example, the RGB color (Red = 0x78, Green = 0xC8, Blue = 0x40) is represented as 0x40C878 in COLORREF notation.

The following methods of the Network class are used to get and set node colors:

Python

```
set_node_bg_color(node: int | str, color: int) -> None
get_node_bg_color(node: int | str) -> int
set_node_text_color(node: int | str, color: int) -> None
get_node_text_color(node: int | str) -> int
set_node_border_color(node: int | str, color: int) -> None
get_node_border_color(node: int | str) -> int
```

Java

```
void setNodeBgColor(int nodeHandle, java.awt.Color color);
void setNodeBgColor(String nodeId, java.awt.Color color);
java.awt.Color getNodeBgColor(int nodeHandle);
java.awt.Color getNodeBgColor(String nodeId);
void setNodeTextColor(int nodeHandle, java.awt.Color color);
void setNodeTextColor(String nodeId, java.awt.Color color);
java.awt.Color getNodeTextColor(int nodeHandle);
java.awt.Color getNodeTextColor(String nodeId);
void setNodeBorderColor(int nodeHandle, java.awt.Color color);
void setNodeBorderColor(String nodeId, java.awt.Color color);
java.awt.Color getNodeBorderColor(int nodeHandle);
java.awt.Color getNodeBorderColor(String nodeId);
```

C#

```
void SetNodeBgColor(int nodeHandle, System.Drawing.Color color);
void SetNodeBgColor(string nodeId, System.Drawing.Color color);
System.Drawing.Color GetNodeBgColor(int nodeHandle);
System.Drawing.Color GetNodeBgColor(string nodeId);
void SetNodeTextColor(int nodeHandle, System.Drawing.Color color);
void SetNodeTextColor(string nodeId, System.Drawing.Color color);
System.Drawing.Color GetNodeTextColor(int nodeHandle);
System.Drawing.Color GetNodeTextColor(string nodeId);
void SetNodeBorderColor(int nodeHandle, System.Drawing.Color color);
void SetNodeBorderColor(string nodeId, System.Drawing.Color color);
System.Drawing.Color GetNodeBorderColor(int nodeHandle);
System.Drawing.Color GetNodeBorderColor(string nodeId);
```

R

```
setNodeBgColor(nodeIdOrHandle, color)
bgColor <- getNodeBgColor(nodeIdOrHandle)
setNodeTextColor(nodeIdOrHandle, color)
textColor <- getNodeTextColor(nodeIdOrHandle)
setNodeBorderColor(nodeIdOrHandle, color)
borderColor <- getNodeBorderColor(nodeIdOrHandle)
```

6.3.3 Arcs

The graph defined by nodes and arcs in a Network instance is always a **directed acyclic graph** (DAG). Any attempt to add an arc that would introduce a cycle will result in an [exception](#) ²⁸.

The following methods of the Network class are used to **add or delete arcs** between nodes:

Python

```
add_arc(parent: int | str, child: int | str) -> None
delete_arc(parent: int | str, child: int | str) -> None
```

Java

```
void addArc(int parentHandle, int childHandle);
void addArc(String parentId, String childId);
void deleteArc(int parentHandle, int childHandle);
void deleteArc(String parentId, String childId);
```

C#

```
void AddArc(int parentHandle, int childHandle);
void AddArc(string parentId, string childId);
void DeleteArc(int parentHandle, int childHandle);
void DeleteArc(string parentId, string childId);
```

R

```
addArc(parentIdOrHandle, childIdOrHandle)
deleteArc(parentIdOrHandle, childIdOrHandle)
```

To **inspect the network structure**, use the following methods. Methods ending with Ids return arrays of string identifiers for the related nodes, while methods without the suffix return arrays of integer node handles..

Python

```
get_parents(node: int | str) -> List[int]
get_parent_ids(node: int | str) -> List[str]
get_children(node: int | str) -> List[int]
get_child_ids(node: int | str) -> List[str]
```

Java

```
int[] getParents(int nodeHandle);
int[] getParents(String nodeId);
String[] getParentIds(int nodeHandle);
String[] getParentIds(String nodeId);
int[] getChildren(int nodeHandle);
int[] getChildren(String nodeId);
String[] getChildIds(int nodeHandle);
String[] getChildIds(String nodeId);
```

C#

```
int[] GetParents(int nodeHandle);
int[] GetParents(string nodeId);
string[] GetParentIds(int nodeHandle);
string[] GetParentIds(string nodeId);
int[] GetChildren(int nodeHandle);
int[] GetChildren(string nodeId);
string[] GetChildIds(int nodeHandle);
string[] GetChildIds(string nodeId);
```

R

```
parentHandles <- getParents(nodeIdOrHandle)
parentIds <- getParentIds(nodeIdOrHandle)
childHandles <- getChildren(nodeIdOrHandle)
childIds <- getChildIds(nodeIdOrHandle)
```

6.4 Multidimensional arrays

Conditional probability table (CPT) describes the interaction between a node and its immediate predecessors. The number of dimensions and the total size of a conditional probability table are determined by the number of parents, the number of states of each of these parents, and the number of states of the child node. Essentially, there is a probability for every state of the child node for every combination of the states of the parents. Nodes that have no predecessors are specified by a prior probability distribution table, which specifies the prior probability of every state (outcome) of the node.

The conditional probability tables are stored as vectors of doubles that are a flattened version of multidimensional tables with as many dimensions as there are parents plus one for the node itself. The order of the coordinates reflects the order in which the arcs to the node were created. The most significant (leftmost) coordinate will represent the state of the first parent. The state of the node itself corresponds to the least significant (rightmost) coordinate.

The image below is an annotated screenshot of GeNIe's node properties window open for the *Forecast* node in the model created in Tutorial 1. *Forecast* has three outcomes and two parents: *Success of the venture* and *State of the economy*, with two and three outcomes, respectively. Therefore, the total size of the CPT is $2 \times 3 \times 3 = 18$. The annotation arrows in the image (not part of the actual GeNIe window) show the ordering of the entries in the linear buffer used internally. The first element (index 0 in SMILE, all indices used by the library are zero-based), the one with the value of 0.7 and yellow background, represents $P(\text{Forecast}=\text{Good} \mid \text{Success of the venture}=\text{Success} \ \& \ \text{State of the economy}=\text{Up})$. It is followed by the probabilities for *Moderate* and *Poor* outcomes given the same parent configuration. The next parent configuration is *Success of the venture*=*Success* & *State of the economy*=*Flat*, and so on.

Success of the venture		Success			Failure		
State of the economy		Up	Flat	Down	Up	Flat	Down
Good		0.7	0.65	0.6	0.15	0.1	0.05
Moderate		0.29	0.3	0.3	0.3	0.3	0.25
Poor		0.01	0.05	0.1	0.55	0.6	0.7

The node definition returned by `Network.get_node_definition` is a single-dimensional array. If your program needs to convert between linear and multidimensional coordinates, see [Tutorial 3](#)¹⁰¹. The code in this tutorial includes the conversion function.

Note that multidimensional arrays are not used exclusively for CPTs. Other uses include expected utility tables and marginal probability distributions.

6.5 Input and output

SMILE supports two types of network I/O: string-based and file-based.

The native format for SMILE networks is XDSL. This format is XML-based, and its definition schema is available on the BayesFusion documentation website (<http://support.bayesfusion.com/docs/>). When a network is saved in this format, the `.xdsl` file extension must be used. XDSL is the only format supported by string-based I/O methods. File-based methods support additional formats; however, depending on the level of feature compatibility between SMILE and third-party software, some information may be lost during conversion.

If an error occurs during an I/O operation (such as when a non-existent file is specified), an [exception](#)²⁸ will be thrown.

The following methods of the `Network` class are used for input and output:

Python

```
read_file(xdsl_file: str) -> None
write_file(xdsl_file: str) -> None
read_string(xdsl_string: str) -> None
write_string() -> str
```

Java

```
void readFile(String fileName)
void writeFile(String fileName)
void readString(String xsdlString)
String writeString()
```

C#

```
void ReadFile(string fileName)
```

```
void WriteFile(string fileName)
void ReadString(sting xds1String)
string WriteString()
```

R

```
readFile(fileName)
writeFile(fileName)
readString(xds1String)
xds1string <- writeString()
```

6.6 Inference

SMILE includes functions for several popular **Bayesian network inference algorithms**, including the clustering algorithm and various approximate stochastic sampling algorithms. To run inference , or switch between algorithm implementations, use the following methods:

Python

```
update_beliefs()
set_bayesian_algorithm(algorithm_type: int) -> None
get_bayesian_algorithm() -> int
set_influence_diagram_algorithm(algorithm_type: int) -> None
get_influence_diagram_algorithm() -> int
get_sample_count() -> int
set_sample_count(sample_count: int) -> None
get_rand_seed() -> int
set_rand_seed(seed: int) -> None
```

Java

```
void updateBeliefs();
void setBayesianAlgorithm(int algorithmType);
int getBayesianAlgorithm();
void setInfluenceDiagramAlgorithm(int algorithmType);
int getSampleCount();
void setSampleCount(int sampleCount);
int getRandSeed();
void setRandSeed(int seed);
```

C#

```
void UpdateBeliefs()
int BayesianAlgorithm { get; set; }
int InfluenceDiagramAlgorithm { get; set; }
int SampleCount { get; set; }
int RandSeed { get; set; }
```

R

```
updateBeliefs()
setBayesianAlgorithm(algorithmType)
algorithmType <- getBayesianAlgorithm()
setInfluenceDiagramAlgorithm(algorithmType)
algorithmType <- getInfluenceDiagramAlgorithm()
sampleCount <- getSampleCount()
setSampleCount(sampleCount)
seed <- getRandSeed()
setRandSeed(seed)
```

The default algorithm for discrete Bayesian networks is clustering over a network preprocessed with relevance. The output of this algorithm is exact, unlike the results from various sampling algorithms. Sampling-based inference is stochastic and relies on a pseudorandom number generator, which by default is seeded from the system clock. To control the quality of approximation, the number of generated samples can be set with the `Network.set_sample_count` method. Naturally, generating more samples improves the approximation but

increases computation time. For repeatable results across multiple runs, the random number generator can be explicitly seeded using `Network.set_rand_seed` with a non-zero value. This seed is stored in the network object and is used to initialize the generator before each inference call. If the seed is non-zero, it is also saved in the XDSL file, ensuring that the stochastic inference can be reproduced consistently when the network is reloaded.

Available Bayesian inference algorithms are:

- LAURITZEN: the default exact clustering algorithm
- HENRION: logic sampling
- PEARL: Pearl's algorithm for polytrees
- L_SAMPLING: likelihood sampling
- SELF_IMPORTANCE: self importance sampling
- HEURISTIC_IMPORTANCE: heuristic importance sampling
- BACK_SAMPLING: backward sampling
- AIS_SAMPLING: adaptive importance sampling
- EPIS_SAMPLING: EPIS sampling (recommended stochastic algorithm)
- LBP: Loopy belief propagation

There are two algorithms for influence diagrams:

- POLICY_EVALUATION: evaluates all decisions, this is the default for influence diagrams
- FIND_BEST_POLICY: finds best decision

For detailed descriptions, theoretical background, and references, see the GeNIe manual.

6.6.1 Target nodes

To run inference efficiently, **simply mark the nodes you are interested in as targets** using `Network.set_target`. Target nodes are always updated by the inference algorithm, while other nodes may or may not be updated, depending on the algorithm. If no target nodes are explicitly specified, SMILE treats all nodes as targets and updates every node in the network. For small and medium-sized models, explicitly setting targets is usually unnecessary. The library automatically applies relevance reasoning, which takes both the target nodes and the evidence in the network into account, reducing the computational effort required to update only the relevant nodes.

`Network.update_beliefs` may throw an [exception](#)^[28] with error code -42 if the temporary data structures required exceed available memory. Marking only the necessary nodes as targets helps reduce memory usage and avoids this problem.

The following methods of the `Network` class allow you to set, query, or clear target nodes:

Python

```
set_target(node: int | str, target: bool) -> None
is_target(node: int | str) -> bool
```

```
clear_all_targets() -> None
```

Java

```
void setTarget(int nodeHandle, boolean target);
void setTarget(String nodeId, boolean target);
boolean isTarget(int nodeHandle);
boolean isTarget(String nodeId);
void clearAllTargets();
```

C#

```
void SetTarget(int nodeHandle, bool target);
void SetTarget(string nodeId, bool target);
bool IsTarget(int nodeHandle);
bool IsTarget(string nodeId);
void ClearAllTargets();
```

R

```
setTarget(nodeIdOrHandle, target)
isTarget <- isTarget(nodeIdOrHandle)
clearAllTargets()
```

6.6.2 Noisy-MAX decomposition

[*Noisy-MAX*](#)^[51] nodes can be handled more efficiently by enabling SMILE's **Noisy-MAX decomposition**. When decomposition is active, the relevance reasoning layer automatically transforms Noisy-MAX nodes into a form that reduces the complexity of inference, often yielding substantial performance improvements. The behavior of the decomposition can also be fine-tuned by setting a limit on the number of parents allowed in the temporary structures created during inference. The following methods let you check whether decomposition is enabled, toggle it on or off, and configure the decomposition limit:

Python

```
is_noisy_decomposition_enabled() -> bool
set_noisy_decomposition_enabled(enable: bool) -> None
get_noisy_decomposition_limit() -> int
set_noisy_decomposition_limit(limit: int) -> None
```

Java

```
boolean isNoisyDecompositionEnabled();
void setNoisyDecompositionEnabled(boolean enable);
int getNoisyDecompositionLimit();
void setNoisyDecompositionLimit(int limit);
```

C#

```
bool IsNoisyDecompositionEnabled { get; set; }
int NoisyDecompositionLimit { get; set; }
```

R

```
enabled <- isNoisyDecompositionEnabled()
setNoisyDecompositionEnabled(enable)
limit <- getNoisyDecompositionLimit()
setNoisyDecompositionLimit(limit)
```

6.6.3 Probability of evidence

To calculate the probability of the current evidence set in the network, call `Network.prob_evidence`. The function returns a floating-point number between 0 and 1 representing $P(e)$, the probability of all evidence currently assigned to nodes.

Python

```
prob_evidence() -> float
```

Java

```
double probEvidence();
```

C#

```
double ProbEvidence()
```

R

```
evidenceProb <- probEvidence()
```

6.6.4 Annealed MAP

The Annealed MAP algorithm (Yuan et al., 2004) solves the problem of finding the most likely configuration of values of a set of nodes given observations of another subset of nodes. This problem is often called **Maximum A-posteriori Probability (MAP)**. The Annealed MAP algorithm is approximate and solves the problem by means of an approximate optimization procedure called simulated annealing. While the solution is approximate, it performs well in practice and it gives an idea of the order of magnitude of the true maximum. The Annealed MAP algorithm drastically extends the class of MAP problems that can be solved.

The `Network.annealed_map` method finds a maximum a-posteriori assignment for a specified set of nodes, **given the evidence currently set in the network**. The `map_nodes` argument identifies the nodes for which the most probable outcomes should be determined. The output includes an array of the most probable outcomes for the nodes in `map_nodes`, with the order of outcomes matching the order of the nodes in the argument list. If all unobserved nodes are included in `map_nodes`, it may be simpler to use the [Most probable explanation \(MPE\)](#)⁴⁴ algorithm instead.

Python

```
annealed_map(map_nodes: List[int] | List[str], tuning: AnnealedMapTuning) -> AnnealedMapResults
```

Java

```
AnnealedMapResults annealedMap(int[] mapNodes, AnnealedMapTuning tuning);
AnnealedMapResults annealedMap(String[] mapNodes, AnnealedMapTuning tuning);
```

C#

```
AnnealedMapResults AnnealedMap(int[] mapNodes, AnnealedMapTuning tuning);
AnnealedMapResults AnnealedMap(string[] mapNodes, AnnealedMapTuning tuning);
```

R

```
results <- annealedMap(mapNodesOrHandles, tuning)
```

6.6.5 Most probable explanation

The **Most probable explanation (MPE)** algorithm in SMILE is a special case of the [Annealed MAP](#)⁴⁴ algorithm and it solves the problem of finding the most likely configuration of values of all model nodes (those nodes that have been observed already have their most probable values) rather than a selected subset of nodes. The Annealed MAP algorithm is approximate and solves the problem by means of an approximate optimization procedure called simulated annealing.

MPE calculation takes into account any **evidence currently set** on nodes in the network.

Python

```
most_probable_explanation() -> pysmile.MpeResults
```

Java

```
MpeResults mostProbableExplanation();
```

C#

```
MpeResults mpeResults = MostProbableExplanation();
```

R

```
mpeResults <- mostProbableExplanation()
```

The MPE results object contains two integer arrays: one with the handles of unobserved nodes and another with the corresponding most probable outcomes for those nodes.

6.7 Case manager

(work in progress)

6.8 User properties

To integrate application-specific data with SMILE, you can use **user properties**. User properties are arrays of **key–value pairs** available both at the network level and at the node level. They are stored in XDSL files and persist across I/O operations.

Each property consists of a name (key) and a value, both represented as strings in the UserProperty object. Property names must be unique within the set of properties defined for a given node or network. They follow the same conventions as SMILE [identifiers](#)^[28]. Property names starting with DSL_ are reserved for internal use.

Python

```
get_user_properties() -> List[UserProperty]
set_user_properties(properties: List[UserProperty]) -> None
get_node_user_properties(node: int | str) -> List[UserProperty]
set_node_user_properties(node: int | str, properties: List[UserProperty]) -> None
```

Java

```
UserProperty[] getUserProperties();
void setUserProperties(UserProperty[] properties);
UserProperty[] getNodeUserProperties(int nodeHandle);
UserProperty[] getNodeUserProperties(String nodeId);
void setNodeUserProperties(int nodeHandle, UserProperty[] properties);
void setNodeUserProperties(String nodeId, UserProperty[] properties);
```

C#

```
UserProperty[] GetUserProperties();
void SetUserProperties(UserProperty[] properties);
UserProperty[] GetNodeUserProperties(int nodeHandle);
UserProperty[] GetNodeUserProperties(string nodeId);
void SetNodeUserProperties(int nodeHandle, UserProperty[] properties);
void SetNodeUserProperties(string nodeId, UserProperty[] properties);
```

R

```
userProperties <- getUserProperties()
setUserProperties(properties)
nodeProperties <- getNodeUserProperties(nodeIdOrHandle)
setNodeUserProperties(nodeIdOrHandle, properties)
```

6.9 Submodels

For user interface purposes and to make complex network structures easier to understand and navigate, nodes can be organized into a hierarchy of **submodels**. Every network contains at least one **main submodel**, which cannot be deleted and serves as the default container for new nodes. Additional submodels can be added or deleted as needed. Each submodel (except the main submodel) has exactly one parent submodel and may have multiple child submodels, allowing for flexible hierarchical organization of nodes. **Assigning a node to a submodel does not affect its behavior in inference calculations.**

Each submodel in a network has both a numeric **handle** and a textual **identifier**. The submodel handle is a non-negative integer that uniquely identifies the submodel within the network, similar to node handles.

However, submodel handles and node handles exist in separate namespaces—this means that a submodel and a node can share the same numeric value without conflict. The textual identifier of a submodel is unique within the network and persistent across network I/O operations. It can be specified when the submodel is created and can be modified later using the corresponding API methods. You can convert between a submodel’s identifier and its handle using the appropriate `Network.get_submodel` method.

All methods that operate on submodels accept either the numeric handle or the textual identifier. Methods using identifiers may involve a string lookup, while methods using handles perform direct access.

When a submodel is **deleted**, its nodes and child submodels are **transferred to the parent submodel**, preserving the structure of the network.

Use the following methods to manage submodels:

Python

```
get_submodel(id: str) -> int
get_main_submodel() -> int
get_main_submodel_id() -> str
get_submodel_count() -> int
get_first_submodel() -> int
get_next_submodel(handle: int) -> int
add_submodel(parent: int | str, id: str) -> int
delete_submodel(submodel: int | str) -> None
set_submodel_id(submodel: int | str, id: str) -> None
get_submodel_id(submodel: int | str) -> str
set_submodel_name(submodel: int | str, name: str) -> None
get_submodel_name(submodel: int | str) -> str
set_submodel_description(submodel: int | str, description: str) -> None
get_submodel_description(submodel: int | str) -> str
set_submodel_of_node(submodel: int | str, node: int | str) -> None
set_submodel_of_submodel(parent: int | str, child: int | str) -> None
get_submodel_of_node(node: int | str) -> int
get_submodel_of_submodel(submodel: int | str) -> int
get_submodel_position(submodel: int | str) -> List[int]
set_submodel_position(submodel: int | str, x: int, y: int, width: int, height: int) -> None
set_submodel_position(submodel: int | str, rect: List[int]) -> None
```

Java

```
int getSubmodel(String id);
int getMainSubmodel();
String getMainSubmodelId();
int getSubmodelCount();
int getFirstSubmodel();
int getNextSubmodel(int handle);
int addSubmodel(int parentHandle, String id);
int addSubmodel(String parentId, String id);
void deleteSubmodel(int handle);
void deleteSubmodel(String id);
void setSubmodelId(int handle, String id);
String getSubmodelId(int handle);
void setSubmodelName(int handle, String name);
void setSubmodelName(String id, String name);
String getSubmodelName(int handle);
String getSubmodelName(String id);
void setSubmodelDescription(int handle, String description);
void setSubmodelDescription(String id, String description);
String getSubmodelDescription(int handle);
String getSubmodelDescription(String id);
void setSubmodelOfNode(int submodelHandle, int nodeHandle);
void setSubmodelOfNode(String submodelId, String nodeId);
```

```

void setSubmodelOfSubmodel(int parentHandle, int childHandle);
void setSubmodelOfSubmodel(String parentId, String childId);
int getSubmodelOfNode(int nodeHandle);
int getSubmodelOfNode(String nodeId);
int getSubmodelOfSubmodel(int submodelHandle);
int getSubmodelOfSubmodel(String submodelId);
java.awt.Rectangle getSubmodelPosition(int submodelHandle);
java.awt.Rectangle getSubmodelPosition(String submodelId);
void setSubmodelPosition(int submodelHandle, int x, int y, int width, int height);
void setSubmodelPosition(String submodelId, int x, int y, int width, int height);
void setSubmodelPosition(int submodelHandle, java.awt.Rectangle rect);
void setSubmodelPosition(String submodelId, java.awt.Rectangle rect);

```

C#

```

int GetSubmodel(string id);
int GetMainSubmodel();
string GetMainSubmodelId();
int GetSubmodelCount();
int GetFirstSubmodel();
int GetNextSubmodel(int handle);
int AddSubmodel(int parentHandle, string id);
int AddSubmodel(string parentId, string id);
void DeleteSubmodel(int handle);
void DeleteSubmodel(string id);
void SetSubmodelId(int handle, string id);
string GetSubmodelId(int handle);
void SetSubmodelName(int handle, string name);
void SetSubmodelName(string id, string name);
string GetSubmodelName(int handle);
string GetSubmodelName(string id);
void SetSubmodelDescription(int handle, string description);
void SetSubmodelDescription(string id, string description);
string GetSubmodelDescription(int handle);
string GetSubmodelDescription(string id);
void SetSubmodelOfNode(int submodelHandle, int nodeHandle);
void SetSubmodelOfNode(string submodelId, string nodeId);
void SetSubmodelOfSubmodel(int parentHandle, int childHandle);
void SetSubmodelOfSubmodel(string parentId, string childId);
int GetSubmodelOfNode(int nodeHandle);
int GetSubmodelOfNode(string nodeId);
int GetSubmodelOfSubmodel(int submodelHandle);
int GetSubmodelOfSubmodel(string submodelId);
System.Drawing.Rectangle GetSubmodelPosition(int submodelHandle);
System.Drawing.Rectangle GetSubmodelPosition(string submodelId);
void SetSubmodelPosition(int submodelHandle, int x, int y, int width, int height);
void SetSubmodelPosition(string submodelId, int x, int y, int width, int height);
void SetSubmodelPosition(int submodelHandle, System.Drawing.Rectangle rect);
void SetSubmodelPosition(string submodelId, System.Drawing.Rectangle rect);

```

R

```

submodelHandle <- getSubmodel(id)
mainSubmodelHandle <- getMainSubmodel()
mainSubmodelId <- getMainSubmodelId()
submodelCount <- getSubmodelCount()
firstSubmodelHandle <- getFirstSubmodel()
nextSubmodelHandle <- getNextSubmodel(handle)
submodelHandle <- addSubmodel(parentIdOrHandle, id)
deleteSubmodel(submodelIdOrHandle)
setSubmodelId(submodelIdOrHandle, id)
submodelId <- getSubmodelId(submodelIdOrHandle)
setSubmodelName(submodelIdOrHandle, name)
submodelName <- getSubmodelName(submodelIdOrHandle)

```

```

setSubmodelDescription(submodelIdOrHandle, description)
submodelDescription <- getSubmodelDescription(submodelIdOrHandle)
setSubmodelOfNode(submodelIdOrHandle, nodeIdOrHandle)
setSubmodelOfSubmodel(parentIdOrHandle, childIdOrHandle)
submodelOfNode <- getSubmodelOfNode(submodelIdOrHandle)
submodelOfSubmodel <- getSubmodelOfSubmodel(submodelIdOrHandle)
submodelPosition <- getSubmodelPosition(submodelIdOrHandle)
setSubmodelPosition(submodelIdOrHandle, x, y, width, height)
setSubmodelPosition(submodelIdOrHandle, rect)

```

6.10 Discrete nodes and numeric domains

By default, discrete nodes represent random variables with categorical outcomes, each identified by a unique string label (the outcome identifier). In SMILE 2, discrete nodes can also model numerical random variables, allowing calculations over numeric outcomes. Each outcome can be associated either with a numeric interval or with a discrete numeric value (point value).

Intervals are used to define **continuous** numeric distributions over the outcomes, while **point values** are used to define **discrete** numeric distributions, where each outcome has a single associated numeric value.

6.10.1 Outcome intervals

Discrete nodes by default do not have any numeric information associated with their outcomes. To define a **continuous distribution over numeric node outcomes**, use `Network.set_intervals`:

Python

```
net.set_intervals(nodeHandle, [ 0, 2.78, 3.14, 77, math.inf ], True)
```

Java

```
net.setIntervals(nodeHandle, new double[] { 0, 2.78, 3.14, 77, Double.POSITIVE_INFINITY }, true);
```

C#

```
net.setIntervals(nodeHandle, new double[] { 0, 2.78, 3.14, 77, Double.PositiveInfinity }, true);
```

R

```
net$setIntervals(nodeHandle, c({ 0, 2.78, 3.14, 77, Inf })), TRUE)
```

The example defines five numeric boundaries, creating four intervals: 0–2.78, 2.78–3.14, 3.14–77, and 77–infinity. Using negative infinity as the first element creates an open first interval. Interval boundaries must be sorted from lowest to highest. A point interval can be defined by using identical numbers as its boundaries.

The second parameter of `Network.set_intervals` indicates whether existing outcome identifiers should be discarded. Using `true` discards identifiers, leaving an empty string for each outcome. Using identifiers with intervals is optional.

Outcome-modifying methods such as `Network.add_outcome`, `Network.insert_outcome`, and `Network.delete_outcome` work normally with intervals. Adding or inserting outcomes splits an existing interval; deleting removes the corresponding interval. At any time, you can redefine all intervals with `Network.set_intervals`. The new list may have a different size than the previous one.

Other interval-related methods include `Network.has_intervals`, `Network.get_intervals`, and `Network.remove_intervals`. Removing intervals restores default outcome identifiers if they were previously empty. The number of outcomes does not change.

After inference, the node value contains the marginal probability distribution over its outcomes. With intervals defined, you can calculate the mean and standard deviation of the distribution. SMILE treats the distribution as continuous and uniform within each closed interval. Open intervals are assumed to follow a half-normal

distribution, scaled to match the neighboring interval's uniform height. Retrieve the mean or standard deviation via `Network.get_node_mean` and `Network.get_node_std_dev` once `is_value_valid` returns true.

Numeric intervals allow continuous evidence in discrete nodes. Set continuous evidence using `Network.set_cont_evidence`. The evidence is internally converted to an outcome index during inference. If a discrete parent with intervals has continuous evidence and a continuous child, the exact evidence value is used during sampling to evaluate the child.

Here are the methods for managing numeric intervals in discrete nodes:

Python

```
has_intervals(node: int | str) -> bool
set_intervals(node: int | str, boundaries: List[float], discard_ids: bool) -> None
get_intervals(node: int | str) -> list[float]
remove_intervals(node: int | str) -> None
```

Java

```
boolean hasIntervals(int nodeHandle);
boolean hasIntervals(String nodeId);
void setIntervals(int nodeHandle, double[] boundaries, boolean removeIds);
void setIntervals(String nodeId, double[] boundaries, boolean removeIds);
double[] getIntervals(int nodeHandle);
double[] getIntervals(String nodeId);
void removeIntervals(int nodeHandle);
void removeIntervals(String nodeId);
```

C#

```
bool HasIntervals(int nodeHandle);
bool HasIntervals(string nodeId);
void SetIntervals(int nodeHandle, double[] boundaries, bool discardIds);
void SetIntervals(string nodeId, double[] boundaries, bool discardIds);
double[] GetIntervals(int nodeHandle);
double[] GetIntervals(string nodeId);
void RemoveIntervals(int nodeHandle);
void RemoveIntervals(string nodeId);
```

R

```
hasIntervals <- has_intervals(nodeIdOrHandle)
setIntervals(nodeIdOrHandle, boundaries, discardIds)
intervals <- get_intervals(nodeIdOrHandle)
remove_intervals(nodeIdOrHandle)
```

6.10.2 Outcome point values

To define a **discrete distribution over numeric node outcomes**, use `Network.set_node_point_values`. The example below assumes that the node has three outcomes:

Python

```
net.set_point_values(nodeHandle, [ 3.14, -2.78, 77 ])
```

Java

```
net.setPointValues(nodeHandle, new double[] { 3.14, -2.78, 77 });
```

C#

```
net.SetPointValues(nodeHandle, new double[] { 3.14, -2.78, 77 });
```

R

```
net$setPointValues(nodeHandle, c({ 3.14, -2.78, 77 })))
```

Each numeric value is associated with an existing node outcome. There is no requirement that the point values be ordered, and outcome identifiers are preserved. Nodes with point values always require outcome identifiers.

Outcome-modifying methods such as `Network.add_outcome`, `Network.insert_outcome`, or `Network.delete_outcome` work normally on nodes with point values. Any newly created outcomes will have their point values initialized to zero. Point values can only be reassigned in bulk, by calling `Network.set_point_values` again.

After successful inference, the value of a discrete node contains the marginal probability distribution over its outcomes. For nodes with point values, the mean and standard deviation of the distribution can be calculated on demand. These can be retrieved using `Network.get_node_mean` and `Network.get_node_std_dev` once `is_value_valid` returns true.

Point values do not make a discrete node equivalent to a continuous node. Continuous evidence cannot be applied to nodes with point values: calling `Network.set_cont_evidence` will fail, even if the evidence value exactly matches one of the defined point values.

To remove point values from a node, call `Network.remove_point_values`. To check whether a node currently has point values defined, call `Network.has_point_values`.

The methods for working with point values are:

Python

```
has_point_values(node: int | str) -> bool
set_point_values(node: int | str, point_values: List[float]) -> None
get_point_values(node: int | str) -> List[float]
remove_point_values(node: int | str) -> None
```

Java

```
boolean hasPointValues(int nodeHandle);
boolean hasPointValues(String nodeId);
void setPointValues(int nodeHandle, double[] pointValues);
void setPointValues(String nodeId, double[] pointValues);
double[] getPointValues(int nodeHandle);
double[] getPointValues(String nodeId);
void removePointValues(int nodeHandle);
void removePointValues(String nodeId);
```

C#

```
bool HasPointValues(int nodeHandle);
bool HasPointValues(string nodeId);
void SetPointValues(int nodeHandle, double[] pointValues);
void SetPointValues(string nodeId, double[] pointValues);
double[] GetPointValues(int nodeHandle);
double[] GetPointValues(string nodeId);
void RemovePointValues(int nodeHandle);
void RemovePointValues(string nodeId);
```

R

```
hasPointValues <- hasPointValues(nodeIdOrHandle)
setPointValues(nodeIdOrHandle, pointValues)
values <- getPointValues(nodeIdOrHandle)
removePointValues(nodeIdOrHandle)
```

6.11 Canonical nodes

Canonical probabilistic nodes—such as Noisy-MAX/OR, Noisy-MIN/AND, and Noisy-Adder gates—implemented in SMILE, are powerful tools for knowledge engineering in practical applications. For a general CPT binary node with n binary parents, the user must specify 2^n parameters. This number grows exponentially with the number of parents: for $n = 10$, the number of parameters is 1,024; for $n = 20$, it is 1,048,576, with each additional parent doubling the total.

In contrast, a Noisy-OR node requires only $n+1$ parameters—one for each parent plus a single “leak” parameter. For example, with $n = 10$ parents, only 11 parameters are needed; with $n = 20$, only 21 parameters. This dramatic reduction in parameters makes canonical nodes easier to specify and maintain, while also implicitly capturing independence relationships that reduce computational complexity. Using canonical gates therefore simplifies both model construction and inference.

To create a canonical node, pass `NOISY_MAX` or `NOISY_ADDER` to `Network.add_node`. Each canonical node type has its own specific attributes accessible through `Network` class methods, but otherwise behaves like a standard CPT node: it has at least two outcomes, and it can be used as a parent or child wherever a CPT node is allowed.

6.11.1 Noisy-MAX

Noisy-MAX is a **generalization** of the widely used canonical gate Noisy-OR and can model interactions among variables with multiple states. When all nodes involved are binary, a Noisy-MAX node reduces to a standard Noisy-OR node. SMILE’s implementation of Noisy-MAX also includes an implicit negation feature. By DeMorgan’s laws, the OR function (or its generalization, MAX) combined with negation can express any logical relationship, including AND (or MIN) functions. This means that SMILE’s Noisy-MAX can be used to model Noisy-AND/MIN gates, as well as more complex logical interactions.

SMILE’s inference algorithm includes a specialized code path for networks containing Noisy-MAX nodes, which can substantially speed up computations. See the [Noisy-MAX decomposition](#)⁴³ section of this manual for further details.

To create a Noisy-MAX node, use the `NOISY_MAX` type with `Network.add_node`:

Python

```
net.add_node(pysmile.NodeType.NOISY_MAX, "node1")
```

Java

```
net.addNode(Network.NodeType.NOISY_MAX, "node1");
```

C#

```
net.AddNode(Network.NodeType.NoisyMax, "node1");
```

R

```
net$addNode(net$NodeType$NOISY_MAX, "node1")
```

A Noisy-MAX node has discrete outcomes, like a standard CPT node. Its node definition includes two key components:

- **Conditionally Independent (CI) Probabilities** – define the distribution of the child node given the states of its parents, assuming conditional independence. Set or retrieve them with `Network.set_node_definition` and `Network.get_node_definition`.
- **Parent Outcome Strengths** – for each parent, an array of numerical strengths specifies the order in which parent states influence the child in the CI table. The Noisy-MAX CI table is always organized according to these strengths. Use `Network.set_noisy_parent_strengths` and `Network.get_noisy_parent_strengths` to modify or read them.

As an example, consider a binary Noisy-MAX node with two parents, each having three outcomes. The following snippet modifies the probabilities and outcome strengths for the second parent (zero-based index 1):

Python

```
ci = net.get_node_definition(h)
BASE = 2 * 3
ci[BASE] = 0.1
ci[BASE + 1] = 0.9
ci[BASE + 2] = 0.3
```

```

ci[BASE + 3] = 0.7
net.set_node_definition(h, ci)
net.set_noisy_parent_strengths(h, 1, [2, 0, 1])

```

Java

```

double[] ci = net.getNodeDefiniton(h);
int BASE = 2 * 3;
ci[BASE] = 0.1;
ci[BASE + 1] = 0.9;
ci[BASE + 2] = 0.3;
ci[BASE + 3] = 0.7;
net.setNodeDefiniton(h, ci);
net.setNoisyParentStrengths(h, 1, new int[] { 2, 0, 1 });

```

C#

```

double[] ci = net.GetNodeDefinition(h);
int BASE = 2 * 3;
ci[BASE] = 0.1;
ci[BASE + 1] = 0.9;
ci[BASE + 2] = 0.3;
ci[BASE + 3] = 0.7;
net.SetNoisyParentStrengths(h, 1, new int[] { 2, 0, 1 });

```

R

```

ci <- net$getNodeDefiniton(h)
BASE <- 2 * 3
ci[BASE + 1] <- 0.1
ci[BASE + 2] <- 0.9
ci[BASE + 3] <- 0.3
ci[BASE + 4] <- 0.7
net$setNodeDefiniton(h, ci)
net$setNoisyParentStrengths(h, 1, c(2L, 0L, 1L))

```

Explanation:

- BASE is calculated as the product of the child node's outcome count and the product of preceding parents' outcome counts. Here, one preceding parent has three outcomes, giving $BASE = 2 * 3$
- Each pair of probabilities in the CI table corresponds to a binary child node for a given parent outcome. The probabilities within a pair always sum to 1.
- The parent outcome strengths array $[2, 0, 1]$ defines the order in which the parent's outcomes are interpreted by the Noisy-MAX child node. For example, after applying the strengths, the second parent's outcomes are seen by the child as $\{State2, State0, State1\}$.
- Changing the strengths **does not modify the parent** node itself; the ordering is local to this parent-child relationship.
- Other Noisy-MAX nodes can define their own parent outcome ordering if they share the same parent.

This structure ensures that probabilities remain valid (sum to 1 for each parent outcome) while allowing flexible control over which parent outcome maps to which column in the child's CI table.

Assuming that we started with the default uniform probabilities in the table, our modifications yields the following Noisy-Max definition, as viewed in GeNIe:

Parent	⊖	p1			⊖	p2		LEAK
State		State0	State1	State2	State2	State0	State1	
► State0		0.5	0.5	0	0.1	0.3	0	0.5
State1		0.5	0.5	1	0.9	0.7	1	0.5

The outcomes of both parent are $\{State0, State1, State2\}$. However, by using `set_parent_outcome_strengths` for parent $p2$, its outcomes are seen by the Noisy-MAX child node as $\{State2, State0, State1\}$. Other Noisy-MAX nodes in the same network can set up their own parent outcome ordering if $p2$ becomes their parent.

The conditionally independent (CI) probabilities of a Noisy-MAX node are used internally to derive the full conditional probability table (CPT) for the node. This expansion is performed automatically by the library whenever the complete CPT is required during inference. The resulting CPT can be retrieved using the `Network.get_noisy_expanded_definition` method.

The following methods of the `Network` class allow you to work with parent outcome strengths and retrieve the expanded CPT for Noisy-MAX nodes:

Python

```
set_noisy_parent_strengths(node: int | str, parent: int | str, strengths: list[int]) -> None
get_noisy_parent_strengths(node: int | str, parent: int | str) -> list[int]
get_noisy_expanded_definition(node: int | str) -> list[float]
```

Java

```
void setNoisyParentStrengths(int nodeHandle, int parentIndex, int[] strengths);
void setNoisyParentStrengths(String nodeId, int parentIndex, int[] strengths);
void setNoisyParentStrengths(int nodeHandle, String parentId, int[] strengths);
void setNoisyParentStrengths(String nodeId, String parentId, int[] strengths);
int[] getNoisyParentStrengths(int nodeHandle, int parentIndex);
int[] getNoisyParentStrengths(String nodeId, int parentIndex);
int[] getNoisyParentStrengths(int nodeHandle, String parentId);
int[] getNoisyParentStrengths(String nodeId, String parentId);
double[] getNoisyExpandedDefinition(int nodeHandle);
double[] getNoisyExpandedDefinition(String nodeId);
```

C#

```
void SetNoisyParentStrengths(int nodeHandle, int parentIndex, int[] strengths);
void SetNoisyParentStrengths(string nodeId, int parentIndex, int[] strengths);
void SetNoisyParentStrengths(int nodeHandle, string parentId, int[] strengths);
void SetNoisyParentStrengths(string nodeId, string parentId, int[] strengths);
int[] GetNoisyParentStrengths(int nodeHandle, int parentIndex);
int[] GetNoisyParentStrengths(string nodeId, int parentIndex);
int[] GetNoisyParentStrengths(int nodeHandle, string parentId);
int[] GetNoisyParentStrengths(string nodeId, string parentId);
double[] GetNoisyExpandedDefinition(int nodeHandle);
double[] GetNoisyExpandedDefinition(string nodeId);
```

R

```
set_noisy_parent_strengths(nodeIdOrHandle, parentIdOrIndex, strengths)
noisy_parent_strengths <- get_noisy_parent_strengths(nodeIdOrHandle, parentIdOrIndex)
noisy_expanded_definition <- get_noisy_expanded_definition(nodeIdOrHandle)
```

6.11.2 Noisy-Adder

Noisy-Adder nodes are a type of canonical node in SMILE, designed to model cumulative effects of parent nodes. Currently, they are not fully supported in the wrappers.

If you need to work with Noisy-Adder nodes, please contact BayesFusion. We can provide a pre-release version of a SMILE wrapper compatible with your programming language and operating system. Keep in mind that functionality may be limited, and API changes could occur in the final release.

6.12 Influence diagrams

Influence diagrams extend Bayesian networks by adding specialized node types to model decision-making:

- **Decision nodes** represent variables under the control of a decision maker, modeling the available alternatives explicitly as the possible outcomes of the node. They have no numerical parameters, only a discrete set of outcomes, and can be children of both decision and chance nodes. To create a decision node, pass `DECISION` to `Network.add_node`.
- **Value nodes** quantify the desirability of outcomes in the decision process. They are specified by the utility associated with each combination of parent node outcomes and can be children of decision and chance nodes. To create a value node, use `UTILITY` with `Network.add_node`.
- **Multi-attribute utility (MAU) nodes** combine multiple value nodes to define a multi-attribute utility function. The function can be specified as a set of weights for a linear function—making the node an additive linear utility (ALU)—or as expressions that reference parent value node identifiers. MAU nodes can be children of decision, value, and other MAU nodes. If decision parents exist, the MAU definition includes a separate set of weights or expressions for each combination of decision parent outcomes. To create a MAU node, pass `MAU` to `Network.add_node`. By default, MAU nodes are defined by weights; to use expressions, call `Network.set_mau_expressions`. For a list of available functions and syntax, see the [Equations reference](#)¹⁸² chapter.

As in Bayesian networks, the results of influence diagram inference are stored in node values and accessed via `Network.get_node_value`. However, interpretation is extended for influence diagrams. The returned matrices are indexed by **indexing parents**, which are unobserved decision nodes that precede the current node or unobserved chance nodes that are predecessors of decision nodes and should have been observed before decisions can be made. Use `Network.get_indexing_parents` to retrieve their handles or `Network.get_indexing_parent_ids` to retrieve their identifiers. The set of outcomes of indexing parents is called a policy.

After successful inference in an influence diagram, node values represent:

- **Chance and deterministic nodes:** posterior **probabilities** for each policy
- **Decision nodes:** expected **utilities** for all outcomes and for each policy
- **Value and MAU nodes:** expected **utility** for each policy

See [Tutorial 4](#)¹⁰⁹ for a simple influence diagram demo program.

6.13 Dynamic Bayesian networks

A Bayesian network is a snapshot of the system at a given time and is used to model systems that are in some kind of equilibrium state. Unfortunately, most systems in the world change over time and sometimes we are interested in how these systems evolve over time more than we are interested in their equilibrium states. Whenever the focus of our reasoning is change of a system over time, we need a tool that is capable of modeling dynamic systems.

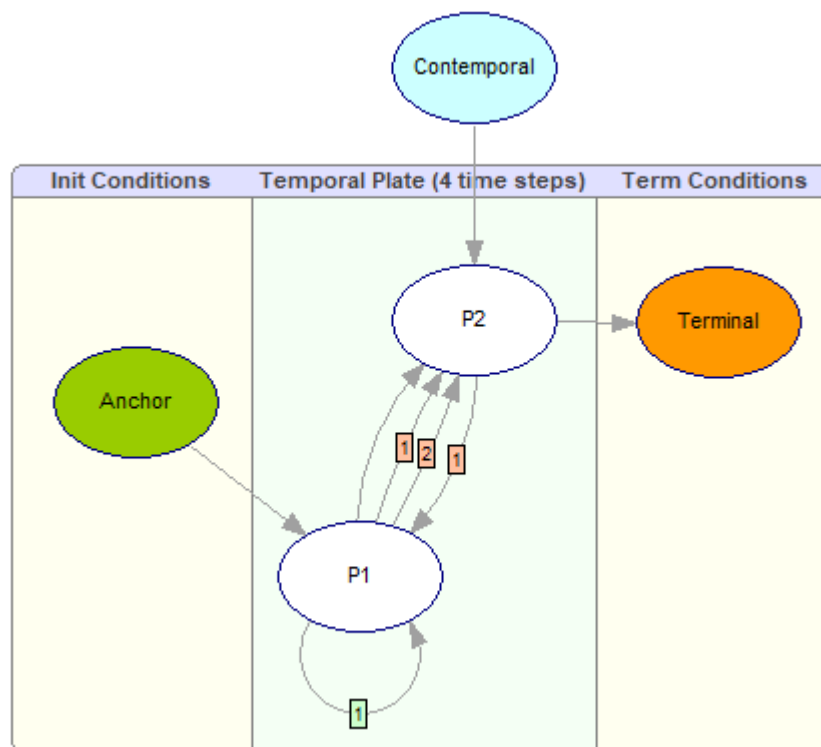
A dynamic Bayesian network (DBN) is a Bayesian network extended with additional mechanisms that are capable of modeling influences over time (Murphy, 2002). The temporal extension of BNs does not mean that the network structure or parameters changes dynamically, but that a dynamic system is modeled. In other words, the underlying process, modeled by a DBN, is stationary. A DBN is a model of a stochastic process.

SMILE's implementation of DBNs supports discrete chance (CPT and canonical), discrete deterministic, and equation nodes in dynamic models.

[Tutorial 6](#)¹²² contains a complete program demonstrating the use of DBNs.

6.13.1 Temporal node types

A **dynamic Bayesian network (DBN)** is visualized in GeNIe with a **temporal plate**. The structure of the DBN shown below (as a GeNIe screenshot) does not represent any real system; it was created solely for demonstration purposes.



The temporal plate divides the graphical layout of the network into four categories of nodes. These categories correspond to four **temporal node types** in SMILE:

- **CONTEMPORAL:** Nodes that do not change over time.
- **ANCHOR:** Nodes outside the temporal plate that have one or more children inside the plate. Anchor nodes are similar to contemporal nodes, but they are only connected to nodes (their children) in the first time slice of the network. In GeNIe, anchor nodes are displayed in the Init Conditions panel on the left-hand side of the temporal plate.
- **PLATE:** Nodes located inside the temporal plate that may have incoming and/or outgoing temporal arcs. They are displayed in the Temporal Plate panel, which also indicates the number of time slices used during inference.
- **TERMINAL:** Nodes outside the temporal plate that have one or more parents inside the plate. Terminal nodes are connected only to nodes (their parents) in the last time slice of the network. They are displayed in the Term Conditions panel on the right-hand side of the temporal plate.

Use `Network.set_node_temporal_type` and `get_node_temporal_type` to modify or retrieve a node's temporal type.

Python

```
get_node_temporal_type(node: int | str) -> int
set_node_temporal_type(node: int | str, temporal_type: int) -> None
```

Java

```
int getNodeTemporalType(int nodeHandle);
int getNodeTemporalType(String nodeId);
void setNodeTemporalType(int nodeHandle, int temporalType);
void setNodeTemporalType(String nodeId, int temporalType);
```

C#

```
int GetNodeTemporalType(int nodeHandle);
int GetNodeTemporalType(string nodeId);
void SetNodeTemporalType(int nodeHandle, int temporalType);
void SetNodeTemporalType(string nodeId, int temporalType);
```

R

```
temporalType <- net$getNodeTemporalType(nodeIdOrHandle)
net$setNodeTemporalType(nodeIdOrHandle, temporalType)
```

6.13.2 Temporal arcs

In dynamic Bayesian networks, **temporal arcs** represent dependencies between variables across different time slices. They define how the state of a variable at time t -*order_of_arc* influences the state of another variable at time t . The parent and child of a temporal arc may even be the same PLATE node, expressing temporal self-dependence (for example, how a variable's past value affects its future value).

Temporal arcs connect only PLATE nodes, since these are the nodes that represent variables evolving over time. Each temporal arc has an associated **temporal order**, indicating how many time steps separate the connected nodes. Temporal arcs of order 1 connect consecutive time slices, while higher-order arcs connect nodes separated by two or more time steps.

In the screenshot included in the [Temporal node types](#)⁵⁵ section, there are three temporal arcs:

- from P1 to P2 with order 1,
- from P1 to P2 with order 2,
- from P2 to P1 with order 1.

Temporal arcs are added with `Network.add_temporal_arc`, which takes the parent and child node identifiers and the desired temporal order as arguments. The arcs are automatically expanded when the DBN is unrolled during inference. Arcs that do not span across time slices are referred to as added using `Network.add_arc`.

The following methods allow you to create, delete, and inspect temporal arcs, as well as retrieve temporal parents, children, and orders for nodes in a dynamic Bayesian network:

Python

```
add_temporal_arc(parent: int | str, child: int | str, order: int) -> None
delete_temporal_arc(parent: int | str, child: int | str, order: int) -> None
temporal_arc_exists(parent: int | str, child: int | str, order: int) -> bool
get_max_temporal_order() -> int
get_max_node_temporal_order(node: int | str) -> int
get_temporal_orders(node: int | str) -> List[int]
get_temporal_children(node: int | str) -> List[TemporalInfo]
get_temporal_parents(node: int | str, order: int) -> List[TemporalInfo]
```

Java

```

void addTemporalArc(int parentHandle, int childHandle, int order);
void addTemporalArc(String parentId, String childId, int order);
void deleteTemporalArc(int parentHandle, int childHandle, int order);
void deleteTemporalArc(String parentId, String childId, int order);
boolean temporalArcExists(int parentHandle, int childHandle, int order);
boolean temporalArcExists(String parentId, String childId, int order);
int getMaxTemporalOrder();
int getMaxNodeTemporalOrder(int nodeHandle);
int getMaxNodeTemporalOrder(String nodeId);
int[] getTemporalOrders(int nodeHandle);
int[] getTemporalOrders(String nodeId);
TemporalInfo[] getTemporalChildren(int nodeHandle);
TemporalInfo[] getTemporalChildren(String nodeId);
TemporalInfo[] getTemporalParents(int nodeHandle, int order);
TemporalInfo[] getTemporalParents(String nodeId, int order);

```

C#

```

void AddTemporalArc(int parentHandle, int childHandle, int order);
void AddTemporalArc(string parentId, string childId, int order);
void DeleteTemporalArc(int parentHandle, int childHandle, int order);
void DeleteTemporalArc(string parentId, string childId, int order);
bool TemporalArcExists(int parentHandle, int childHandle, int order);
bool TemporalArcExists(string parentId, string childId, int order);
int GetMaxTemporalOrder();
int GetMaxNodeTemporalOrder(int nodeHandle);
int GetMaxNodeTemporalOrder(string nodeId);
int[] GetTemporalOrders(int nodeHandle);
int[] GetTemporalOrders(string nodeId);
TemporalInfo[] GetTemporalChildren(int nodeHandle);
TemporalInfo[] GetTemporalChildren(string nodeId);
TemporalInfo[] GetTemporalParents(int nodeHandle, int order);
TemporalInfo[] GetTemporalParents(string nodeId, int order);

```

R

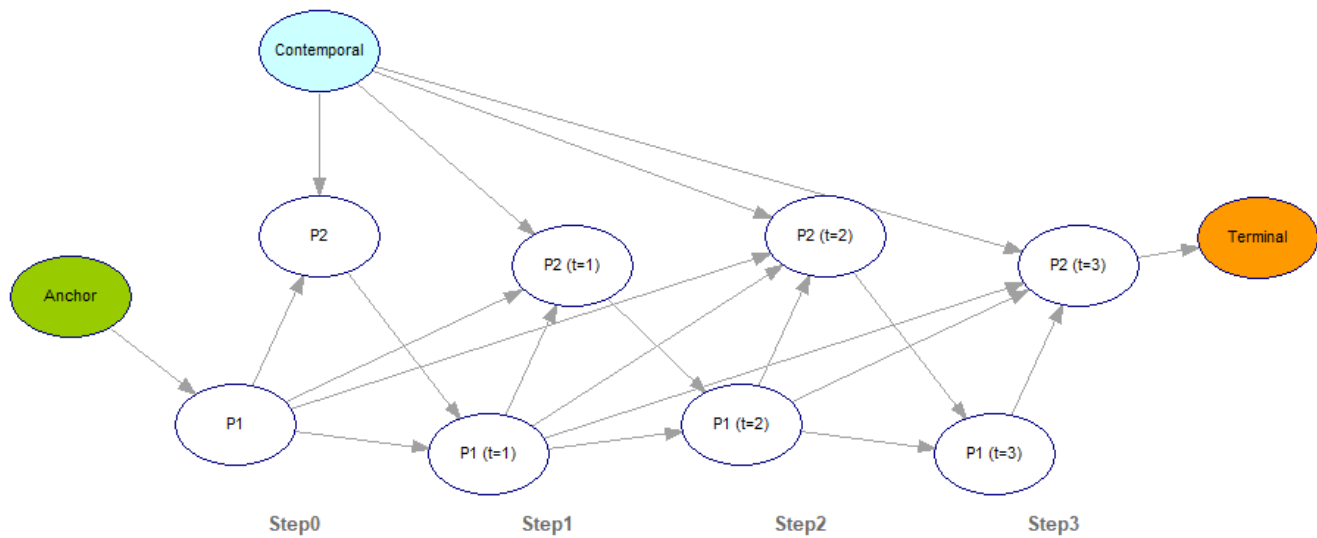
```

addTemporalArc(parentIdOrHandle, childIdOrHandle, order)
deleteTemporalArc(parentIdOrHandle, childIdOrHandle, order)
temporalExists <- temporalArcExists(parentIdOrHandle, childIdOrHandle, order)
maxOrder <- getMaxTemporalOrder()
maxNodeOrder <- getMaxNodeTemporalOrder(nodeIdOrHandle)
temporalOrders <- getTemporalOrders(nodeIdOrHandle)
temporalChildren <- getTemporalChildren(nodeIdOrHandle)
temporalParents <- getTemporalParents(nodeIdOrHandle, order)

```

6.13.3 Unrolling

SMILE's inference algorithm for dynamic Bayesian networks (DBNs) converts—or **unrolls**—a DBN over time into a temporary static network, which is then solved using standard inference algorithm. The results of this inference are automatically copied back into the node values of the original DBN. Below is how our example DBN appears after unrolling:



To reduce the size of the unrolled network, we reduced the number of **time steps (slices)** from the default value of 10 to 4 using `Network.set_slice_count`. The colors of the original nodes are preserved in the unrolled model, which is extended by creating new copies of the plate nodes. The structure of this network illustrates how temporal arcs express the conditional dependency of plate nodes at time step i on plate nodes at time step j , where $i > j$.

Consider the temporal arc between $P1$ and $P2$ with temporal order 2. This arc is copied twice in the unrolled network: once from $P1$ at time slice 0 to $P2$ at time slice 2, and once from $P1$ at time slice 1 to $P2$ at time slice 3. There is no copy starting from $P1$ at time slice 2 because its child would fall at time slice 4, which exceeds the zero-based slice range of 0 to 3 in our example. In contrast, the three temporal arcs with order 1 are copied three times each. The arc linking $P1$ to itself illustrates how self-dependencies propagate: it appears from $P1$ at time slice 0 to $P1$ at time slice 1, from $P1$ at time slice 1 to $P1$ at time slice 2, and from $P1$ at time slice 2 to $P1$ at time slice 3.

Arcs originating from *Contemporal* and *Anchor* nodes behave differently in the unrolled network. The *Anchor* node has children only in the initial time slice, reflecting its role in initializing the dynamic system. The *Contemporal* node, on the other hand, is linked to all time slices, so its children are replicated across each slice. *Terminal* node has parents only in the last slice, consistent with its function of receiving information from the end of the temporal process.

To ensure that the unrolled network remains a directed acyclic graph (DAG), the following arcs are forbidden:

- From plate nodes to normal or anchor nodes.
- From terminal nodes to any non-terminal nodes.

Unrolling is performed automatically during inference. For debugging or explanatory purposes, it is also possible to explicitly obtain an unrolled network by calling `Network.unroll`. The unrolled network created in this way is completely independent of the original DBN: any changes made to the original network after the unroll call do not affect the unrolled model, and modifications to the unrolled network do not propagate back to the original DBN.

Python

```
unroll() -> UnrollResults
get_slice_count() -> int
set_slice_count(slice_count: int) -> None
```

Java

```
UnrollResults unroll();
int getSliceCount();
void setSliceCount(int sliceCount);
```

C#

```
UnrollResults Unroll();
int SliceCount { get; set; }
```

R

```
unrollResults <- unroll()
sliceCount <- getSliceCount()
setSliceCount(sliceCount)
```

6.13.4 Temporal definitions

Consider node P_2 from the previous example. It has four incoming arcs in total:

- a normal arc from a *Contemporal* node
- a normal arc from P_1
- two temporal arcs from P_1 with temporal orders 1 and 2

However, in the unrolled network, the slice for $t=0$ shows P_2 with only two incoming arcs, because there are no slices representing timepoints before $t=0$. In the slice for $t=1$, P_2 has three incoming arcs, as it is now possible to link P_1 at $t=0$ with P_2 at $t=1$. Starting with the slice for $t=2$, P_2 has all four incoming arcs.

This behavior means that P_2 requires **separate CPTs** for $t=0$, $t=1$, and $t \geq 2$. In general, if a plate node has an incoming arc of order x , it will require $x+1$ separate **temporal definitions**. To access or modify these definitions, use `Network.get_node_temporal_definition` and `Network.set_node_temporal_definition`.

All parents contributing to a temporal definition with a specified order can be retrieved using `Network.get_unrolled_parents`. Note that `Network.get_temporal_parents`, called with the same temporal order, returns only a subset of these parents. This distinction arises because unrolling creates additional arcs: in the unrolled network, P_2 has four incoming arcs for slices $t \geq 2$, but only two of these correspond to arcs with temporal order 2.

The following methods of the `Network` class allow you to retrieve and modify temporal definitions for plate nodes, and to inspect the parents that contribute to these definitions in the unrolled network.

Python

```
get_node_temporal_definition(node: int | str, order: int) -> List[float]
set_node_temporal_definition(node: int | str, order: int, definition: List[float]) -> None
get_noisy_temporal_parent_strengths(node: int | str, order: int) -> List[List[int]]
set_noisy_temporal_parent_strengths(node: int | str, order: int, strengths: List[List[int]]) -> None
get_node_temporal_equations(node: int | str) -> List[str]
set_node_temporal_equations(node: int | str, equations: List[str]) -> None
get_unrolled_parents(node: int | str, order: int) -> List[TemporalInfo]
get_unrolled_parents(node: int | str) -> List[TemporalInfo]
```

Java

```
double[] getNodeTemporalDefinition(int nodeHandle, int order);
double[] getNodeTemporalDefinition(String nodeId, int order);
void setNodeTemporalDefinition(int nodeHandle, int order, double[] definition);
void setNodeTemporalDefinition(String nodeId, int order, double[] definition);
int[][] getNoisyTemporalParentStrengths(int nodeHandle, int order);
int[][] getNoisyTemporalParentStrengths(String nodeId, int order);
void setNoisyTemporalParentStrengths(int nodeHandle, int order, int[][] strengths);
```

```

void setNoisyTemporalParentStrengths(String nodeId, int order, int[][] strengths);
String[] getNodeTemporalEquations(int nodeHandle);
String[] getNodeTemporalEquations(String nodeId);
void setNodeTemporalEquations(int nodeHandle, String[] equations);
void setNodeTemporalEquations(String nodeId, String[] equations);
TemporalInfo[] getUnrolledParents(int nodeHandle, int order);
TemporalInfo[] getUnrolledParents(String nodeHandle, int order);
TemporalInfo[] getUnrolledParents(int nodeHandle);
TemporalInfo[] getUnrolledParents(String nodeHandle);

```

C#

```

double[] GetNodeTemporalDefinition(int nodeHandle, int order);
double[] GetNodeTemporalDefinition(string nodeId, int order);
void SetNodeTemporalDefinition(int nodeHandle, int order, double[] definition);
void SetNodeTemporalDefinition(string nodeId, int order, double[] definition);
int[][] GetNoisyTemporalParentStrengths(int nodeHandle, int order);
int[][] GetNoisyTemporalParentStrengths(string nodeId, int order);
void SetNoisyTemporalParentStrengths(int nodeHandle, int order, int[][] strengths);
void SetNoisyTemporalParentStrengths(string nodeId, int order, int[][] strengths);
string[] GetNodeTemporalEquations(int nodeHandle);
string[] GetNodeTemporalEquations(string nodeId);
void SetNodeTemporalEquations(int nodeHandle, string[] equations);
void SetNodeTemporalEquations(string nodeId, string[] equations);
TemporalInfo[] GetUnrolledParents(int nodeHandle, int order);
TemporalInfo[] GetUnrolledParents(string nodeId, int order);
TemporalInfo[] GetUnrolledParents(int nodeHandle);
TemporalInfo[] GetUnrolledParents(string nodeId);

```

R

```

temporal_definition <- getNodeTemporalDefinition(nodeIdOrHandle, order)
setNodeTemporalDefinition(nodeIdOrHandle, order, definition)
noisy_parent_strengths <- getNoisyTemporalParentStrengths(nodeIdOrHandle, order)
setNoisyTemporalParentStrengths(nodeIdOrHandle, order, strengths)
temporalEquations <- getNodeTemporalEquations(nodeIdOrHandle)
setNodeTemporalEquations(nodeIdOrHandle, equations)
unrolledParents <- getUnrolledParents(nodeIdOrHandle, order)
unrolledParents <- getUnrolledParents(nodeIdOrHandle)

```

6.13.5 Temporal evidence

In dynamic Bayesian networks, evidence for plate nodes can be set separately for each time slice. This **temporal evidence** can take several forms: standard discrete evidence, virtual evidence (a probability distribution over outcomes), or continuous evidence. Continuous evidence is allowed when the plate node has numeric intervals defined or when it is an equation-based node. Each type of evidence is stored per time slice and incorporated into inference accordingly.

For time slice 0, normal evidence-related methods can be used as with regular nodes. For slices $t \geq 1$, separate methods are provided to set and retrieve evidence for the specific time slice. The available methods for each programming language are listed below.

Python

```

get_temporal_evidence(node: int | str, slice: int) -> int
get_temporal_evidence_id(node: int | str, slice: int) -> str
get_temporal_virtual_evidence(node: int | str, slice: int) -> List[float]
get_temporal_cont_evidence(node: int | str, slice: int) -> float
set_temporal_evidence(node: int | str, slice: int, outcome_index_or_id: int | str) -> None
set_temporal_virtual_evidence(node: int | str, slice: int, evidence: List[float]) -> None
set_temporal_cont_evidence(node: int | str, slice: int, evidence: float) -> None
clear_temporal_evidence(node: int | str, slice: int) -> None
has_temporal_evidence(node: int | str) -> bool

```

```
is_temporal_evidence(node: int | str, slice: int) -> bool
is_temporal_virtual_evidence(node: int | str, slice: int) -> bool
is_temporal_cont_evidence(node: int | str, slice: int) -> bool
```

Java

```
int getTemporalEvidence(int nodeHandle, int slice);
int getTemporalEvidence(String nodeId, int slice);
String getTemporalEvidenceId(int nodeHandle, int slice);
String getTemporalEvidenceId(String nodeId, int slice);
double[] getTemporalVirtualEvidence(int nodeHandle, int slice);
double[] getTemporalVirtualEvidence(String nodeId, int slice);
double getTemporalContEvidence(int nodeHandle, int slice);
double getTemporalContEvidence(String nodeId, int slice);
void setTemporalEvidence(int nodeHandle, int slice, int outcomeIndex);
void setTemporalEvidence(String nodeId, int slice, int outcomeIndex);
void setTemporalEvidence(int nodeHandle, int slice, String outcomeId);
void setTemporalEvidence(String nodeId, int slice, String outcomeId);
void setTemporalVirtualEvidence(int nodeHandle, int slice, double[] evidence);
void setTemporalVirtualEvidence(String nodeId, int slice, double[] evidence);
void setTemporalContEvidence(int nodeHandle, int slice, double evidence);
void setTemporalContEvidence(String nodeId, int slice, double evidence);
void clearTemporalEvidence(int nodeHandle, int slice);
void clearTemporalEvidence(String nodeId, int slice);
boolean hasTemporalEvidence(int nodeHandle);
boolean hasTemporalEvidence(String nodeId);
boolean isTemporalEvidence(int nodeHandle, int slice);
boolean isTemporalEvidence(String nodeId, int slice);
boolean isTemporalVirtualEvidence(int nodeHandle, int slice);
boolean isTemporalVirtualEvidence(String nodeId, int slice);
boolean isTemporalContEvidence(int nodeHandle, int slice);
boolean isTemporalContEvidence(String nodeId, int slice);
```

C#

```
int GetTemporalEvidence(int nodeHandle, int slice);
int GetTemporalEvidence(string nodeId, int slice);
string GetTemporalEvidenceId(int nodeHandle, int slice);
string GetTemporalEvidenceId(string nodeId, int slice);
double[] GetTemporalVirtualEvidence(int nodeHandle, int slice);
double[] GetTemporalVirtualEvidence(string nodeId, int slice);
double GetTemporalContEvidence(int nodeHandle, int slice);
double GetTemporalContEvidence(string nodeId, int slice);
void SetTemporalEvidence(int nodeHandle, int slice, int outcomeIndex);
void SetTemporalEvidence(string nodeId, int slice, int outcomeIndex);
void SetTemporalEvidence(int nodeHandle, int slice, string outcomeId);
void SetTemporalEvidence(string nodeId, int slice, string outcomeId);
void SetTemporalVirtualEvidence(int nodeHandle, int slice, double[] evidence);
void SetTemporalVirtualEvidence(string nodeId, int slice, double[] evidence);
void SetTemporalContEvidence(int nodeHandle, int slice, double evidence);
void SetTemporalContEvidence(string nodeId, int slice, double evidence);
void ClearTemporalEvidence(int nodeHandle, int slice);
void ClearTemporalEvidence(string nodeId, int slice);
bool HasTemporalEvidence(int nodeHandle);
bool HasTemporalEvidence(string nodeId);
bool IsTemporalEvidence(int nodeHandle, int slice);
bool IsTemporalEvidence(string nodeId, int slice);
bool IsTemporalVirtualEvidence(int nodeHandle, int slice);
bool IsTemporalVirtualEvidence(string nodeId, int slice);
bool IsTemporalContEvidence(int nodeHandle, int slice);
bool IsTemporalContEvidence(string nodeId, int slice);
```

R

```
temporalEvidence <- getTemporalEvidence(nodeIdOrHandle, slice)
temporalEvidenceId <- getTemporalEvidenceId(nodeIdOrHandle, slice)
```

```

temporalVirtualEvidence <- getTemporalVirtualEvidence(nodeIdOrHandle, slice)
temporalContEvidence <- getTemporalContEvidence(nodeIdOrHandle, slice)
setTemporalEvidence(nodeIdOrHandle, slice, outcomeIndexOrId)
setTemporalVirtualEvidence(nodeIdOrHandle, slice, evidence)
setTemporalContEvidence(nodeIdOrHandle, slice, evidence)
clearTemporalEvidence(nodeIdOrHandle, slice)
hasTemporalEvidence <- hasTemporalEvidence(nodeIdOrHandle)
isTemporalEvidence <- isTemporalEvidence(nodeIdOrHandle, slice)
isTemporalVirtualEvidence <- isTemporalVirtualEvidence(nodeIdOrHandle, slice)
isTemporalContEvidence <- isTemporalContEvidence(nodeIdOrHandle, slice)

```

6.13.6 Temporal beliefs

For plate nodes in a dynamic Bayesian network, the inference algorithm computes temporal beliefs, which are marginal posterior probability distributions that vary over time. Temporal beliefs are returned by `Network.get_node_value`.

Because temporal beliefs depend on time, the resulting belief array is larger than for a static node. If a node has X outcomes and the number of unrolled time slices is Y , the array will contain $X * Y$ elements. The elements corresponding to a single time slice are stored contiguously: indices $[0 .. X-1]$ represent beliefs for $t=0$, indices $[X .. 2X-1]$ for $t = 1$, and so on.

Only PLATE nodes contain temporal beliefs. Other temporal node types (such as ANCHOR, TERMINAL, and CONTEMPORAL nodes) have regular belief arrays, whose size equals the number of their outcomes.

6.14 Continuous models

Graphical models, such as Bayesian networks, are not limited to discrete variables. They are, in fact, closely related to systems of simultaneous structural equations. SMILE allows the construction of models that include equation nodes, which serve as alternative, graphical representations of such systems of simultaneous equations.

[Tutorial 7](#)¹³² contains a complete program demonstrating the use of continuous models.

6.14.1 Equation-based nodes

To create an **equation node**, specify `EQUATION` when calling `Network.add_node`. By default, the node equation is set to $x=0$ (where x is the node identifier). You can replace this default equation using `Network.set_node_equation`. For instance, the following code sets the node x to a standard Gaussian probability distribution:

Python

```

hx = net.add_node(pysmile.NodeType.EQUATION, "x")
net.set_node_equation(hx, "x=Normal(0,1)")

```

Java

```

int hx = net.addNode(Network.NodeType.EQUATION, "x");
net.setNodeEquation(hx, "x=Normal(0,1)");

```

C#

```

int hx = net.AddNode(Network.NodeType.Equation, "x");
net.SetNodeEquation(hx, "x=Normal(0,1)");

```

R

```

hx <- net$addNode(net$NodeType$EQUATION, "x")
net$setNodeEquation(hx, "x=Normal(0,1)")

```

SMILE defines many functions for use in node equations. The complete list of functions is available in the [Equations reference](#)¹⁸² section. Among these functions, there are random generators, which generate a single sample based on the passed parameters. In the example above, the *Normal* is the name of SMILE's random generator function.

Node equations can reference other nodes by using their identifiers. Continuing with our code snippet:

Python

```
hy = net.add_node(pysmile.NodeType.EQUATION, "y")
net.set_node_equation(hy, "y=2*x")
```

Java

```
int hy = net.addNode(Network.NodeType.EQUATION, "y");
net.setNodeEquation(hx, "y=2*x");
```

C#

```
int hy = net.AddNode(Network.NodeType.Equation, "y");
net.SetNodeEquation(hx, "y=2*x");
```

R

```
hy <- net$addNode(net$NodeType$EQUATION, "y");
net$setNodeEquation(hx, "y=2*x");
```

A second node is added, and its equation is set to $y=2x$, where x references the previously defined node. SMILE automatically adds an arc from x to y ; it is not necessary to call `Network.add_arc` before setting an equation that references other nodes. Note that calling `addArc` explicitly will modify the child node's equation by appending a term for the new parent at the end of the equation. For example, in a network with equation nodes a , b , c , and d , if d 's equation is set to $d=Normal(a, 1)+Normal(b, 2)$, calling `addArc` with the handles of nodes c and d would rewrite d 's equation as $d=Normal(a, 1)+Normal(b, 2)+c$.

If an arc is removed—either via `Network.delete_arc` or `Network.delete_node` on one of the parents—the node equation is automatically rewritten to sum over the remaining parents. This ensures that equations and arcs remain synchronized. For instance, if node b is deleted or an arc from b to d is removed, d 's equation would update to $d=a+c$.

Changes to node identifiers are also propagated into equations. If the identifier of the first node in the previous example were changed from x to x_{prime} , the equation of node y would automatically update to $y=2x_{prime}$.

In SMILE, calling `Network.set_node_equation` is sufficient to define both the mathematical relationship and the network structure for continuous (equation) nodes. Unlike discrete nodes, where the conditional probability distribution is specified with `Network.set_node_definition`, equation nodes automatically create or update arcs and dependencies based on the expression passed to `set_node_equation`. In this sense, `set_node_equation` can be seen as a generalization of `set_node_definition`, combining parameter specification and structural updates in a single operation.

In addition to `set_node_equation`, there are functions that provide information about the type of an equation defining the node. `Network.get_node_equation_constant` returns the numeric value of a constant equation, if the node represents a constant. The corresponding `Network.is_node_equation_constant` function returns true if the node's equation is a fixed constant, and false otherwise. Similarly, `Network.is_node_equation_deterministic` indicates whether the node's equation is deterministic—that is, whether it produces a single, predictable value given its parent nodes, rather than generating random samples.

Python

```
get_node_equation(node: int | str) -> str
set_node_equation(node: int | str, equation: str) -> None
get_node_equation_constant(node: int | str) -> float
is_node_equation_constant(node: int | str) -> bool
is_node_equation_deterministic(node: int | str) -> bool
```

Java

```
String getNodeEquation(int nodeHandle);
String getNodeEquation(String nodeId);
void setNodeEquation(int nodeHandle, String equation);
void setNodeEquation(String nodeId, String equation);
double getNodeEquationConstant(int nodeHandle);
double getNodeEquationConstant(String nodeId);
boolean isNodeEquationConstant(int nodeHandle);
boolean isNodeEquationConstant(String nodeId);
boolean isNodeEquationDeterministic(int nodeHandle);
boolean isNodeEquationDeterministic(String nodeId);
```

C#

```
string GetNodeEquation(int nodeHandle);
string GetNodeEquation(string nodeId);
void SetNodeEquation(int nodeHandle, string equation);
void SetNodeEquation(string nodeId, string equation);
double GetNodeEquationConstant(int nodeHandle);
double GetNodeEquationConstant(string nodeId);
bool IsNodeEquationConstant(int nodeHandle);
bool IsNodeEquationConstant(string nodeId);
bool IsNodeEquationDeterministic(int nodeHandle);
bool IsNodeEquationDeterministic(string nodeId);
```

R

```
nodeEquation <- getNodeEquation(nodeIdOrHandle)
setNodeEquation(nodeIdOrHandle, equation)
nodeConstant <- getNodeEquationConstant(nodeIdOrHandle)
isConstant <- isNodeEquationConstant(nodeIdOrHandle)
isDeterministic <- isNodeEquationDeterministic(nodeIdOrHandle)
```

6.14.2 Continuous evidence

To set evidence on an equation node, use `Network.set_cont_evidence`. In the example below, `evidenceNodeHandle` refers to the handle of the equation node:

Python

```
net.set_cont_evidence(evidenceNodeHandle, 1.5)
```

Java

```
net.setContEvidence(evidenceNodeHandle, 1.5);
```

C#

```
net.SetContEvidence(evidenceNodeHandle, 1.5);
```

R

```
net$setContEvidence(evidenceNodeHandle, 1.5)
```

To retrieve the current evidence from a continuous node, use `Network.get_cont_evidence`.

Python

```
get_cont_evidence(node: int | str) -> float
set_cont_evidence(node: int | str, value: float) -> None
```

Java

```
double getContEvidence(int nodeHandle);
double getContEvidence(String nodeId);
void setContEvidence(int nodeHandle, double value);
void setContEvidence(String nodeId, double value);
```

C#

```
double GetContEvidence(int nodeHandle);
double GetContEvidence(string nodeId);
```

```
void SetContEvidence(int nodeHandle, double value);
void SetContEvidence(string nodeId, double value);
```

R

```
contEvidenceValue <- getContEvidence(nodeIdOrHandle)
setContEvidence(nodeIdOrHandle, value)
```

6.14.3 Continuous inference

To run inference in a continuous (equation-based) model, use `Network.update_beliefs`, the same method employed for discrete networks.

Inference in continuous networks is performed using stochastic sampling when there is no evidence in the network, or when evidence is present only in nodes without parents, or when evidence is provided in nodes with parents but all parents also have evidence. In all other cases, inference is performed on a temporary discrete network derived from the original continuous model. The definitions of the temporary discrete nodes are obtained from the discretization intervals specified for each continuous node.

Both types of inference algorithms use the lower and upper bounds defined for each equation node. To specify these bounds, use the `Network.set_node_equation_bounds` method. During stochastic sampling, a sample may be rejected if its value falls outside the defined bounds. This behavior can be controlled with `Network.set_outlier_rejection_enabled`; by default, outlier rejection is disabled.

Stochastic inference in continuous networks is controlled using the same mechanisms as in discrete networks. Specifically, the number of samples generated by the stochastic algorithm can be adjusted with `Network.set_sample_count`, and repeatable results can be obtained by setting the pseudorandom generator seed with `Network.set_rand_seed`. These methods allow consistent control over the quality and reproducibility of the sampling-based approximation of the continuous model.

The discretization algorithm is controlled by `Network.set_discretization_sample_count`. Using a larger number of samples generally improves the approximation of the solution to the system of equations represented by the continuous network, but requires more computation time. Additionally, the `Network.set_zero_avoidance_enabled` method can be used to enable zero avoidance. When zero avoidance is enabled, the algorithm avoids placing zeros in the CPTs, even if no samples are generated for a given parameter. Zeros in probability distributions can cause theoretical issues and should be used with caution—only when it is certain that the probability should be zero. Once a zero is set, that probability cannot be changed, regardless of the strength of the evidence against it.

To define a node's **discretization intervals**, use `Network.set_node_equation_discretization`. This method accepts an array of `DiscretizationInterval` objects as its argument. Each interval is defined by an optional identifier (not used for inference) and an upper bound. The lower bound of an interval with index j is taken from the upper bound of the preceding interval (index $j-1$). The lower bound of the first interval is defined by the node's lower bound, which is set with `Network.set_node_equation_bounds`.

Discretization intervals are used to generate CPTs for the temporary discrete network. Each CPT column is calculated by drawing a number of samples specified at the network level. The size of the resulting discretized CPT is the product of the number of intervals of the node and the number of intervals of its parent nodes. These CPTs are used internally during inference to approximate the continuous model. For performance reasons, the discretized CPTs are cached and reused in subsequent inference calls, as long as there are no relevant changes in the network (such as modifications to node equations or discretization intervals). Note that these cached CPTs are not stored in the XDSL file. Be aware that high memory usage may occur when a discretized equation node has many parents.

The results of inference in a continuous model can be either **samples** or **discretized beliefs**, depending on the inference algorithm used:

- **Sampling algorithm** – Produces a set of samples for each node. `Network.get_node_value` returns an array of these samples, and `Network.get_node_sample_stats` provides simple statistics for the sample set (mean, standard deviation, minimum, and maximum). You can also retrieve the mean and standard deviation directly with `Network.get_node_mean` and `Network.get_node_std_dev`. `Network.is_value_discretized` returns false.
- **Discretizing algorithm** – Produces a probability distribution over the discretization intervals. `Network.get_node_value` returns this distribution, and `Network.is_value_discretized` returns true.

The following methods provide access to inference results, evidence, and discretization settings for continuous (equation-based) nodes in SMILE.

Python

```
is_outlier_rejection_enabled() -> bool
set_outlier_rejection_enabled(enabled: bool) -> None
get_discretization_sample_count() -> int
set_discretization_sample_count(discretization_sample_count: int) -> None
is_zero_avoidance_enabled() -> bool
set_zero_avoidance_enabled(enabled: bool) -> None
set_node_equation_bounds(node: int | str, lo: float, hi: float) -> None
get_node_equation_bounds(node: int | str) -> List[float]
set_node_equation_discretization(node: int | str, intervals: List[DiscretizationInterval]) -> None
set_node_equation_uniform_discretization(node: int | str, interval_count: int) -> None
get_node_equation_discretization(node: int | str) -> List[DiscretizationInterval]
is_value_discretized(node: int | str) -> bool
get_node_sample_stats(node: int | str) -> List[float]
get_node_mean(node: int | str) -> float
get_node_stddev(node: int | str) -> float
```

Java

```
boolean isOutlierRejectionEnabled();
void setOutlierRejectionEnabled(boolean enabled);
int getDiscretizationSampleCount();
void setDiscretizationSampleCount(int discretizationSampleCount);
boolean isZeroAvoidanceEnabled();
void setZeroAvoidanceEnabled(boolean enable);
void setNodeEquationBounds(int nodeHandle, double lo, double hi);
void setNodeEquationBounds(String nodeId, double lo, double hi);
double[] getNodeEquationBounds(int nodeHandle);
double[] getNodeEquationBounds(String nodeId);
void setNodeEquationDiscretization(int nodeHandle, DiscretizationInterval[] intervals);
void setNodeEquationDiscretization(String nodeId, DiscretizationInterval[] intervals);
void setNodeEquationUniformDiscretization(int nodeHandle, int intervalCount);
void setNodeEquationUniformDiscretization(String nodeId, int intervalCount);
DiscretizationInterval[] getNodeEquationDiscretization(int nodeHandle);
DiscretizationInterval[] getNodeEquationDiscretization(String nodeId);
boolean isValueDiscretized(int nodeHandle);
boolean isValueDiscretized(String nodeId);
double[] getNodeSampleStats(int nodeHandle);
double[] getNodeSampleStats(String nodeId);
double getNodeMean(int nodeHandle);
double getNodeMean(String nodeId);
double getNodeStdDev(int nodeHandle);
double getNodeStdDev(String nodeId);
```

C#

```
bool OutlierRejectionEnabled { get; set; }
int DiscretizationSampleCount { get; set; }
bool ZeroAvoidanceEnabled { get; set; }
void SetNodeEquationBounds(int nodeHandle, double lo, double hi);
void SetNodeEquationBounds(string nodeId, double lo, double hi);
```

```

double[] GetNodeEquationBounds(int nodeHandle);
double[] GetNodeEquationBounds(string nodeId);
void SetNodeEquationDiscretization(int nodeHandle, DiscretizationInterval[] intervals);
void SetNodeEquationDiscretization(string nodeId, DiscretizationInterval[] intervals);
void SetNodeEquationUniformDiscretization(int nodeHandle, int intervalCount);
void SetNodeEquationUniformDiscretization(string nodeId, int intervalCount);
DiscretizationInterval[] GetNodeEquationDiscretization(int nodeHandle);
DiscretizationInterval[] GetNodeEquationDiscretization(string nodeId);
bool IsValueDiscretized(int nodeHandle);
bool IsValueDiscretized(string nodeId);
double[] GetNodeSampleStats(int nodeHandle);
double[] GetNodeSampleStats(string nodeId);
double GetNodeMean(int nodeHandle);
double GetNodeMean(string nodeId);
double GetNodeStdDev(int nodeHandle);
double GetNodeStdDev(string nodeId);

```

R

```

outlierEnabled <- isOutlierRejectionEnabled()
setOutlierRejectionEnabled(enabled)
discSampleCount <- getDiscretizationSampleCount()
setDiscretizationSampleCount(discSampleCount)
zeroAvoidanceEnabled <- isZeroAvoidanceEnabled()
setZeroAvoidanceEnabled(enabled)
setNodeEquationBounds(nodeIdOrHandle, lo, hi)
bounds <- getNodeEquationBounds(nodeIdOrHandle)
setNodeEquationDiscretization(nodeIdOrHandle, intervals)
setNodeEquationUniformDiscretization(nodeIdOrHandle, intervalCount)
intervals <- getNodeEquationDiscretization(nodeIdOrHandle)
valueDiscretized <- isValueDiscretized(nodeIdOrHandle)
sampleStats <- getNodeSampleStats(nodeIdOrHandle)
meanValue <- getNodeMean(nodeIdOrHandle)
stdDev <- getNodeStdDev(nodeIdOrHandle)

```

6.15 Hybrid models

Hybrid models are Bayesian networks that contain **both discrete and continuous nodes**. Arcs in such networks can connect any combination of node types, meaning that both discrete-to-continuous and continuous-to-discrete relationships are supported. Inference in hybrid models follows the same principles as in continuous networks: stochastic sampling is used when there is no evidence in nodes that have unobserved parents, while discretization is applied otherwise. Conceptually, hybrid models can be viewed as continuous models in which discrete nodes serve as specialized functions—namely, conditional probability tables (CPTs) defined by arrays of numeric values.

Arcs from discrete to continuous nodes are created by referencing the discrete node's identifier in the equation of the continuous node, just as with arcs between continuous nodes. The following discussion assumes familiarity with the functions used to build node equations, which are described in detail in the [Equations reference](#)¹⁸² section of this manual.

For example, suppose node *c* is continuous and node *d* is discrete. The equation for *c* might be written as: $c = \text{Normal}(\text{If}(d=1, 1, -1), 5)$. When evaluating a sample for *c*, one of its inputs will be the value of its parent node *d*. Discrete node values are integers from 0 to $N-1$, where N is the number of outcomes of the discrete node. In this example, the equation specifies that *c* should be drawn from a normal distribution with a standard deviation of 5, while the mean depends on the state of *d*: if *d* is in its state with zero-based index 1, the mean will be 1; otherwise, the mean will be -1.

To improve readability, parent discrete node outcomes can be represented as text literals. For example, if *d* has outcomes *High*, *Medium*, and *Low*, the previous equation can be written as $c = \text{Normal}(\text{If}(d = \text{"Medium"}, 1, -1), 5)$.

Common functions for discrete parents include *Switch* and *Choose*. For instance, $c = \text{Normal}(\text{Switch}(d, \text{"High"}, 3.2, \text{"Medium"}, 2.5, \text{"Low"}, 1.4), 5)$ or equivalently $c = \text{Normal}(\text{Choose}(d, 3.2, 2.5, 1.4), 5)$. These expressions let the mean of c 's normal distribution depend on the state of d .

Alternatively, the ternary operator $?:$ can be used to represent conditional expressions. For example, $c = \text{Normal}(d = \text{"Medium"} ? 1 : -1, 5)$ has the same effect as the *If* function, producing a mean of 1 when d is *Medium*, and -1 otherwise.

Important: SMILE does not modify text literals representing the outcomes of discrete parent nodes if the outcome identifiers are changed. If a text literal in an equation cannot be associated with any outcome of the parent node, its value is evaluated as -1.

In general, discrete nodes can appear anywhere in an equation where numerical values are allowed. For example, $c = \log(1+d)$ or $c = 2^d$ are valid expressions. These equations do not rely on text literals for discrete outcomes, as no comparison to specific outcomes is performed during evaluation.

To add an arc from a continuous node to a discrete node, use `Network.add_arc`, the same method used in purely discrete networks. In this case, the discretization intervals of the continuous parent node are treated as the equivalent of discrete outcomes, allowing the continuous value to influence the discrete child node appropriately.

[Tutorial 8](#)¹⁴² contains a complete program demonstrating the use of hybrid models.

6.16 Diagnosis

Diagnosis is one of the most successful applications of Bayesian networks. The ability of probabilistic knowledge representation techniques to perform both predictive and diagnostic inference makes them particularly suitable for diagnostic tasks. Bayesian networks can combine, or *fuse*, various sources of information—such as patient or equipment history, risk factors, symptoms, and test results—into a coherent probabilistic framework.

A Bayesian network built with SMILE can represent different components of a system, potential faults (or disorders), their observable manifestations (symptoms or alarms), and the results of diagnostic tests. In essence, such a model captures how possible defects—whether in a natural system such as the human body or in an engineered system such as a car, aircraft, or copier—may produce observable effects like error messages, abnormal readings, or test outcomes.

To perform diagnostic analysis, certain nodes in the network must be designated as **diagnostic faults**, while others must be marked as **diagnostic observations**. Optionally, observation nodes can also be assigned **observation costs**, representing, for example, the financial or time cost of performing a medical test or measurement.

The output of SMILE's diagnostic algorithms is twofold:

1. **Posterior marginal probabilities** of diagnostic faults (reflecting their likelihoods given the available evidence).
2. **A ranking of potential observations**, ordered from most to least informative with respect to the current diagnostic focus.

Depending on the selected diagnostic algorithm, the observation ranking is based on one of the following measures:

- **Cross-entropy** – an information-theoretic, utility-free measure that estimates the expected reduction in entropy (uncertainty) over a specified subset of fault states after observing a particular node X . It quantifies the expected value of an observation for improving diagnostic certainty.

- **Distance metrics** – measures of difference between two probability vectors representing the pursued fault states before and after observation. SMILE supports several distance metrics, including Euclidean, cosine, and city-block (Manhattan) distances.

[Tutorial 9](#)¹⁵² demonstrates diagnostic reasoning using the *HeparII* medical diagnosis model, illustrating both single- and multi-fault diagnosis scenarios.

6.16.1 Diagnostic roles

In SMILE, each node participating in diagnosis must be assigned a diagnostic role. This role determines how the node behaves during a diagnostic session. Node roles are set using the `Network.set_node_diag_type` method, and each node can serve as one of the following types:

- **Fault nodes**, identified by `NodeDiagType.FAULT`, represent potential faults or disorders. A fault node must have at least one designated *fault state*. During a diagnostic session, SMILE calculates the posterior probabilities of these fault states as new observations are introduced. To mark a specific state of a diagnostic fault node as a fault state, use `Network.set_fault_outcome`.
- **Observation nodes**, identified by `NodeDiagType.OBSERVATION`, represent measurable findings or test results that can be instantiated during a diagnostic session. A *default outcome* can be assigned to each observation node using `Network.set_default_outcome`; this outcome is automatically instantiated when a diagnostic session begins. To include an observation in the ranked list of diagnostic suggestions, call `Network.set_ranked`. If an observation must be instantiated before ranking can take place, mark it as *mandatory* with `Network.set_mandatory`. Diagnostic algorithms will not produce ranked observation lists until all mandatory observations are instantiated.
- **Auxiliary nodes** identified by `NodeDiagType.AUXILIARY`, are nodes that are neither faults nor observations. These nodes serve as intermediate or supporting variables in the model. The auxiliary role is assigned by default when a node is created.

Diagnostic roles and all related attributes, such as fault outcomes, default outcomes, and ranking or mandatory flags, are stored within the XDSL file, ensuring that diagnostic configurations persist across sessions. The following functions provide access to these diagnostic properties in different programming languages.

Python

```
get_node_diag_type(node: int | str) -> int
set_node_diag_type(node: int | str, type: int) -> None
is_fault_outcome(node: int | str, outcome: int | str) -> bool
set_fault_outcome(node: int | str, outcome: int | str, fault: bool) -> None
get_default_outcome(node: int | str) -> int
get_default_outcome_id(node: int | str) -> str
set_default_outcome(node: int | str, def_outcome: int | str) -> None
is_ranked(node: int | str) -> bool
set_ranked(node: int | str, ranked: bool) -> None
is_mandatory(node: int | str) -> bool
set_mandatory(node: int | str, mandatory: bool) -> None
```

Java

```
int getNodeDiagType(int nodeHandle);
int getNodeDiagType(String nodeId);
void setNodeDiagType(int nodeHandle, int type);
void setNodeDiagType(String nodeId, int type);
boolean isFaultOutcome(int nodeHandle, int outcomeIndex);
boolean isFaultOutcome(String nodeId, int outcomeIndex);
boolean isFaultOutcome(int nodeHandle, String outcomeId);
boolean isFaultOutcome(String nodeId, String outcomeId);
void setFaultOutcome(int nodeHandle, int outcomeIndex, boolean fault);
```

```

void setFaultOutcome(String nodeId, int outcomeIndex, boolean fault);
void setFaultOutcome(int nodeHandle, String outcomeId, boolean fault);
void setFaultOutcome(String nodeId, String outcomeId, boolean fault);
int getDefaultOutcome(int nodeHandle);
int getDefaultOutcome(String nodeId);
String getDefaultOutcomeId(int nodeHandle);
String getDefaultOutcomeId(String nodeId);
void setDefaultOutcome(int nodeHandle, int defOutcome);
void setDefaultOutcome(String nodeId, int defOutcome);
void setDefaultOutcome(int nodeHandle, String defOutcomeId);
void setDefaultOutcome(String nodeId, String defOutcomeId);
boolean isRanked(int nodeHandle);
boolean isRanked(String nodeId);
void setRanked(int nodeHandle, boolean ranked);
void setRanked(String nodeId, boolean ranked);
boolean isMandatory(int nodeHandle);
boolean isMandatory(String nodeId);
void setMandatory(int nodeHandle, boolean mandatory);
void setMandatory(String nodeId, boolean mandatory);

```

C#

```

int GetNodeDiagType(int nodeHandle);
int GetNodeDiagType(string nodeId);
void SetNodeDiagType(int nodeHandle, int type);
void SetNodeDiagType(string nodeId, int type);
bool IsFaultOutcome(int nodeHandle, int outcomeIndex);
bool IsFaultOutcome(string nodeId, int outcomeIndex);
bool IsFaultOutcome(int nodeHandle, string outcomeId);
bool IsFaultOutcome(string nodeId, string outcomeId);
void SetFaultOutcome(int nodeHandle, int outcomeIndex, bool fault);
void SetFaultOutcome(string nodeId, int outcomeIndex, bool fault);
void SetFaultOutcome(int nodeHandle, string outcomeId, bool fault);
void SetFaultOutcome(string nodeId, string outcomeId, bool fault);
int GetDefaultOutcome(int nodeHandle);
int GetDefaultOutcome(string nodeId);
string GetDefaultOutcomeId(int nodeHandle);
string GetDefaultOutcomeId(string nodeId);
void SetDefaultOutcome(int nodeHandle, int defOutcome);
void SetDefaultOutcome(string nodeId, int defOutcome);
void SetDefaultOutcome(int nodeHandle, string defOutcomeId);
void SetDefaultOutcome(string nodeId, string defOutcomeId);
bool IsRanked(int nodeHandle);
bool IsRanked(string nodeId);
void SetRanked(int nodeHandle, bool ranked);
void SetRanked(string nodeId, bool ranked);
bool IsMandatory(int nodeHandle);
bool IsMandatory(string nodeId);
void SetMandatory(int nodeHandle, bool mandatory);
void SetMandatory(string nodeId, bool mandatory);

```

R

```

nodeDiagType <- getNodeDiagType(nodeIdOrHandle)
setNodeDiagType(nodeIdOrHandle, type)
isFault <- isFaultOutcome(nodeIdOrHandle, outcomeIndexOrId)
setFaultOutcome(nodeIdOrHandle, outcomeIndexOrId, fault)
defaultOutcome <- getDefaultOutcome(nodeIdOrHandle)
defaultOutcomeId <- getDefaultOutcomeId(nodeIdOrHandle)
setDefaultOutcome(nodeIdOrHandle, defOutcomeIndexOrId)
ranked <- isRanked(nodeIdOrHandle)
setRanked(nodeIdOrHandle, ranked)
mandatory <- isMandatory(nodeIdOrHandle)
setMandatory(nodeIdOrHandle, mandatory)

```

6.16.2 Observation cost

Observing the value of a variable is often associated with a **cost**. For example, measuring a patient's platelet count requires drawing a blood sample and analyzing it, while measuring the temperature of an air conditioner exhaust unit requires a technician's time. Optimal diagnosis must consider both the value of information and the cost of obtaining it.

For any diagnostic observation node, the cost of observing its value can be specified using `Network.set_node_cost` and retrieved with `Network.get_node_cost`. Costs can be expressed on any scale, such as currency or time in minutes. Negative values are also allowed; these indicate tests that are sufficiently inexpensive that they should always be performed—for example, checking the car model when diagnosing a vehicle.

When the cost of observing a node is independent of other observations, it is specified as a single number. In some cases, however, the cost depends on whether other nodes have been observed. For instance, once a blood sample is drawn, additional tests on the same sample are cheaper than if no sample exists. Similarly, measuring a parameter of a locomotive engine may cost less when the engine is in the shop compared to in the field. Removing a locomotive cover may take hours, but once removed, many tests become inexpensive.

To model such dependencies, use `Network.add_cost_arc`. This adds an **observation cost arc** to the network. Observation cost arcs are independent of standard network arcs. When a diagnostic observation node has incoming cost arcs, its observation cost is represented as a [multidimensional array](#)³⁹, analogous to a conditional probability table, with the last dimension size equal to one.

Cost arcs can be removed using `Network.remove_cost_arc`. The methods `Network.get_cost_parents`, `Network.get_cost_parent_ids` and `Network.get_cost_children` / `Network.get_cost_children_ids` can be used to retrieve information about cost parents and children.

The following methods provide programmatic access to the observation cost of diagnostic nodes.

Python

```
add_cost_arc(parent: int | str, child: int | str) -> None
delete_cost_arc(parent: int | str, child: int | str) -> None
get_cost_parents(node: int | str) -> List[int]
get_cost_parent_ids(node: int | str) -> List[str]
get_cost_children(node: int | str) -> List[int]
get_cost_child_ids(node: int | str) -> List[str]
get_node_cost(node: int | str) -> List[float]
set_node_cost(node: int | str, cost: List[float]) -> None
```

Java

```
void addCostArc(int parentHandle, int childHandle);
void addCostArc(String parentId, String childId);
void deleteCostArc(int parentHandle, int childHandle);
void deleteCostArc(String parentId, String childId);
int[] getCostParents(int nodeHandle);
int[] getCostParents(String nodeId);
String[] getCostParentIds(int nodeHandle);
String[] getCostParentIds(String nodeId);
int[] getCostChildren(int nodeHandle);
int[] getCostChildren(String nodeId);
String[] getCostChildIds(int nodeHandle);
String[] getCostChildIds(String nodeId);
double[] getNodeCost(int nodeHandle);
double[] getNodeCost(String nodeId);
void setNodeCost(int nodeHandle, double[] cost);
void setNodeCost(String nodeId, double[] cost);
```

C#

```

void AddCostArc(int parentHandle, int childHandle);
void AddCostArc(string parentId, string childId);
void DeleteCostArc(int parentHandle, int childHandle);
void DeleteCostArc(string parentId, string childId);
int[] GetCostParents(int nodeHandle);
int[] GetCostParents(string nodeId);
string[] GetCostParentIds(int nodeHandle);
string[] GetCostParentIds(string nodeId);
int[] GetCostChildren(int nodeHandle);
int[] GetCostChildren(string nodeId);
string[] GetCostChildIds(int nodeHandle);
string[] GetCostChildIds(string nodeId);
double[] GetNodeCost(int nodeHandle);
double[] GetNodeCost(string nodeId);
void SetNodeCost(int nodeHandle, double[] cost);
void SetNodeCost(string nodeId, double[] cost);

```

R

```

add_cost_arc(parentIdOrHandle, childIdOrHandle)
delete_cost_arc(parentIdOrHandle, childIdOrHandle)
costParentHandles <- get_cost_parents(nodeIdOrHandle)
costParentIds <- get_cost_parent_ids(nodeIdOrHandle)
costChildHandles <- get_cost_children(nodeIdOrHandle)
costChildIds <- get_cost_child_ids(nodeIdOrHandle)
nodeCcost <- get_node_cost(nodeIdOrHandle)
set_node_cost(nodeIdOrHandle, cost)

```

6.16.3 Diagnostic session

With diagnostic attributes—such as node roles and, optionally, observation costs—defined, a diagnostic session can be initiated to generate a ranked list of candidate observations. The **diagnostic session** is represented by a `DiagNetwork` object. Its constructor requires a reference to the `Network` object where the diagnostic information is defined.

The `DiagNetwork` instance provides an API for pursuing faults, instantiating observations, and retrieving observation statistics. During a diagnostic session, all interactions with the diagnostic model should be performed through the `DiagNetwork` instance.

Python

```
diagnet = pysmile.DiagNetwork(net)
```

Java

```
DiagNetwork diagNet = new DiagNetwork(net);
```

C#

```
DiagNetwork diagNet = new DiagNetwork(net);
```

R

```
diagNet <- DiagNetwork(net)
```

The diagnostic session automatically instantiates mandatory observations during initialization. To obtain a ranking of uninstantiated observations, their diagnostic value must be calculated. This value is dynamic, depending on the observations already instantiated and the fault or faults selected as the focus of reasoning, referred to as **pursued faults**. By default, the pursued fault is the most likely fault.

From the diagnostic session perspective, a fault is represented as a node/state pair, since a single fault node may have multiple faulty states. Node/state pairs are referenced by their indices; fault node handles should not be used directly when specifying the pursued fault. To convert a node handle/state pair to a fault index, use

`DiagNetwork.get_fault_index`. To change the pursued fault, use `DiagNetwork.set_pursued_fault`, or `DiagNetwork.set_pursued_faults` when pursuing multiple faults simultaneously.

During the diagnostic session, observation nodes can be set to evidence using `DiagNetwork.instantiate_observation`. The `Network.setEvidence` method **should not be called directly** during a session, as it can render the internal session state inconsistent. Instantiated observations can be reset to unobserved using `DiagNetwork.release_observation`, which internally calls `Network.clearEvidence`.

After selecting the pursued faults and instantiating known observations, the application can call `DiagNetwork.update` to obtain a `DiagResults` object, which contains two arrays:

- `DiagResults.faults` contains an entry for each fault node in the network. Each entry includes the probability of the fault state in its `probability` member. The array is sorted in descending order of fault probability.
- `DiagResults.observations` contains an entry for each uninstantiated observation node in the network. Each entry includes the diagnostic measure in its `measure` member, and the `info_gain` member contains the effective measure combined with the observation cost. If no cost is defined, `info_gain` is equal to `measure`. The array is sorted in descending order of `info_gain`.

A typical diagnostic program is interactive, beginning with a single pursued fault and no instantiated observations other than those with default values. The user may change the pursued faults and instantiate observations based on the diagnostic values displayed in the interface.

The following methods of the `DiagNetwork` class provide programmatic access to the diagnostic session, allowing pursued faults to be set, observations to be instantiated, and diagnostic results to be retrieved.

Python

```
update() -> DiagResults
restart() -> None
set_pursued_fault(faultIndex: int) -> None
set_pursued_faults(faultIndices: List[int]) -> None
get_pursued_fault() -> int
get_pursued_faults() -> List[int]
get_fault_count() -> int
get_unperformed_test_count() -> int
instantiate_observation(node: int | str, outcome: int | str) -> None
release_observation(node: int | str) -> None
get_unperformed_observations() -> List[int]
get_unperformed_observation_ids() -> List[str]
mandatories_instantiated() -> bool
find_most_likely_fault() -> int
get_fault_index(node: int | str, outcome: int | str) -> int
get_fault_node(faultIndex: int) -> int
get_fault_node_id(faultIndex: int) -> str
get_fault_outcome(faultIndex: int) -> int
get_fault_outcome_id(faultIndex: int) -> str
```

Java

```
DiagResults update();
void restart();
void setPursuedFault(int faultIndex);
void setPursuedFaults(int[] faultIndices);
int getPursuedFault();
int[] getPursuedFaults();
int getFaultCount();
int getUnperformedTestCount();
void instantiateObservation(int nodeHandle, int outcomeIndex);
```

```

void instantiateObservation(int nodeHandle, String outcomeId);
void instantiateObservation(String nodeId, String outcomeId);
void instantiateObservation(String nodeId, int outcomeIndex);
void releaseObservation(int nodeHandle);
void releaseObservation(String nodeId);
int[] getUnperformedObservations();
String[] getUnperformedObservationIds();
boolean mandatoriesInstantiated();
int findMostLikelyFault();
int getFaultIndex(int nodeHandle, int outcomeIndex);
int getFaultIndex(int nodeHandle, String outcomeId);
int getFaultIndex(String nodeId, String outcomeId);
int getFaultIndex(String nodeId, int outcomeIndex);
int getFaultNode(int faultIndex);
String getFaultNodeId(int faultIndex);
int getFaultOutcome(int faultIndex);
String getFaultOutcomeId(int faultIndex);

```

C#

```

DiagResults Update();
void Restart();
void SetPursuedFault(int faultIndex);
void SetPursuedFaults(int[] faultIndices);
int GetPursuedFault();
int[] GetPursuedFaults();
int GetFaultCount();
int GetUnperformedTestCount();
void InstantiateObservation(int nodeHandle, int outcomeIndex);
void InstantiateObservation(int nodeHandle, string outcomeId);
void InstantiateObservation(string nodeId, int outcomeIndex);
void InstantiateObservation(string nodeId, string outcomeId);
void ReleaseObservation(int nodeHandle);
void ReleaseObservation(string nodeId);
int[] GetUnperformedObservations();
string[] GetUnperformedObservationIds();
bool MandatoriesInstantiated();
int FindMostLikelyFault();
int GetFaultIndex(int nodeHandle, int outcomeIndex);
int GetFaultIndex(int nodeHandle, string outcomeId);
int GetFaultIndex(string nodeId, string outcomeId);
int GetFaultIndex(string nodeId, int outcomeIndex);
int GetFaultNode(int faultIndex);
string GetFaultNodeId(int faultIndex);
int GetFaultOutcome(int faultIndex);
string GetFaultOutcomeId(int faultIndex);

```

R

```

diagResults <- update()
restart()
pursuedFault <- get_pursued_fault()
pursuedFaults <- get_pursued_faults()
set_pursued_fault(faultIndex)
set_pursued_faults(faultIndices)
faultCount <- get_fault_count()
unperformedTestCount <- get_unperformed_test_count()
instantiate_observation(nodeIdOrHandle, outcomeIndexOrId)
release_observation(nodeIdOrHandle)
unperformedObservations <- get_unperformed_observations()
unperformedObservationIds <- get_unperformed_observation_ids()
mandatoriesInstantiated <- mandatories_instantiated()
mostLikelyFault <- find_most_likely_fault()
faultIndex <- get_fault_index(nodeIdOrHandle, outcomeIndexOrId)

```

```

faultNode <- get_fault_node(faultIndex)
faultNodeId <- get_fault_node_id(faultIndex)
faultOutcome <- get_fault_outcome(faultIndex)
faultOutcomeId <- get_fault_outcome_id(faultIndex)

```

6.16.4 Distance and entropy-based measures

The **diagnostic measures** calculated for observations depend on the pursued fault set. Pursued faults define the focus of reasoning, and changes in their probabilities serve as inputs to the measure algorithms. To select the algorithm, use `DiagNetwork.set_single_fault_algorithm` and `DiagNetwork.set_multi_fault_algorithm`. The output of each algorithm is a single number for every uninstantiated observation. The identifiers of the algorithms are defined in `DiagNetwork` class.

Single-fault diagnosis algorithms:

- **Max probability change**, identified by `SingleFaultAlgorithmType.PROB_CHANGE` (default): For each observation, the maximum change is taken over the absolute values of the probability changes of the pursued fault. This is a signed measure; values may be negative when the largest-magnitude change corresponds to a probability decrease.
- **Cross-entropy**, identified by `SingleFaultAlgorithmType.CROSSENTROPY`: An information-theoretic, unsigned measure that considers both the information content of individual observation states and the likelihood of observing them. For example, a positive cancer test may cause a large probability change, but if the test is unlikely in a generally healthy person, the expected diagnostic value is small.
- **Normalized cross-entropy** identified by `SingleFaultAlgorithmType.NORM_CROSSENTROPY`: The cross-entropy divided by the current entropy of the pursued fault node.

Multi-fault diagnosis algorithms:

- **Max probability change**, identified by `MultiFaultAlgorithmType.MAX_PROB_CHANGE` (default): The measure value is the maximum probability change over all pursued faults and observation outcomes. This is a signed measure.
- **Euclidean distance (L2 norm)**, identified by `MultiFaultAlgorithmType.L2_NORMALIZED_DISTANCE`: Computes the Euclidean distance between fault probability vectors before and after the observation. Distances are normalized so that a change from impossible (all probabilities zero) to certain (all probabilities one) equals 1.0. The maximum distance over all outcomes is selected for each observation. Larger distances indicate a greater impact.
- **Cityblock distance**, identified by `MultiFaultAlgorithmType.CITYBLOCK_DISTANCE`: Similar to L2, but using the cityblock (Manhattan) metric.
- **Averaged L2 and cityblock distance**, identified by `MultiFaultAlgorithmType.AVG_L2_CITY_DISTANCE`: Combines the two distance measures.
- **Cosine distance (cosine similarity)**, identified by `MultiFaultAlgorithmType.COSINE_DISTANCE`: Calculated between two fault probability vectors. Probabilities are non-negative, so the measure is always non-negative.
- **Entropy-based measures**: A family of six entropy-based measures, which require calculation of the joint probability distribution over all pursued faults, which is computationally prohibitive, it is necessary to use approximations of the joint probability distribution. The approximations are based on two strong assumptions about dependencies among them: (1) complete independence (this is taken by the first group of approaches)

and (2) complete dependence (this is taken by the second group of approaches). Each of the two extremes is divided into three groups: (1) At Least One, (2) Only One, and (3) All. These refer to different partitioning of the combinations of faults in cross-entropy calculation. The identifiers of the algorithms are:

- `MultiFaultAlgorithmType.INDEPENDENCE_AT_LEAST_ONE`
- `MultiFaultAlgorithmType.INDEPENDENCE_ONLY_ONE`
- `MultiFaultAlgorithmType.INDEPENDENCE_ONLY_ALL`
- `MultiFaultAlgorithmType.DEPENDENCE_AT_LEAST_ONE`
- `MultiFaultAlgorithmType.DEPENDENCE_AT_LEAST_ONEf`
- `MultiFaultAlgorithmType.DEPENDENCE_AT_LEAST_ONEf`
- **Marginal probability-based measures:** Much faster than the independence/dependence-based joint probability distribution approaches but it is not as accurate because they make a stronger assumption about the joint probability distribution. Entropy calculations in this approach are based purely on the marginal probabilities of the pursued faults. The two algorithms that use the Marginal Probability Approach differ essentially in the function that they use to select the tests to perform. Both functions are scaled so that they return values between 0 and 1. Entropy/Marginal 1 uses a function without the support for maximum distance and its minimum is reached when all probabilities of the faults are equal to 0.5. Entropy/Marginal 2 uses a function that has support for maximum distance and is continuous in the domain [0,1]. The identifiers of the algorithms are `MultiFaultAlgorithmType.MARGINAL_1` and `MultiFaultAlgorithmType.MARGINAL_2`.

6.17 Learning

Learning in SMILE supports two primary tasks:

- **Structure learning:** Create a new network from a dataset.
- **Parameter learning:** Refine the parameters (conditional probability tables) of an existing network.

SMILE also supports **network validation**, which is commonly performed after learning to evaluate the quality and reliability of the resulting network.

Both learning and validation use data stored in a `DataSet` object. The `DataSet` class provides a table-like container for variable values used during training and evaluation. Its content is typically loaded from a comma-separated values (CSV) file or a similar tabular format. A `DataSet` can also be populated programmatically, for example by iterating over database query results. This is an application-specific task handled by the user's program.

6.17.1 Learning network structure

The following classes can be used to learn Network from `DataSet`:

- **BayesianSearch:** performs structure learning using a hill-climbing procedure guided by a scoring heuristic with random restarts.
- **NaiveBayes:** implements the standard Naive Bayes structure learning method.
- **TAN:** Tree-Augmented Naive Bayes, a semi-naive method based on the Bayesian Search approach.

- ABN: Augmented Naive Bayes, another semi-naive method derived from Bayesian Search.

In the simplest scenario, create an instance of the desired learning algorithm class and call its learn method to perform network learning.

Python

```
ds = pysmile.learning.DataSet()
ds.read_file("mydatafile.txt")
baySearch = pysmile.learning.BayesianSearch()
net = baySearch.learn(ds)
```

Java

```
DataSet ds = new DataSet();
ds.readFile("mydatafile.txt");
BayesianSearch baySearch = new BayesianSearch();
Network net = baySearch.learn(ds);
```

C#

```
DataSet ds = new DataSet();
ds.ReadFile("mydatafile.txt");
BayesianSearch baySearch = new BayesianSearch();
Network net = baySearch.Learn(ds);
```

R

```
ds <- DataSet()
ds$readFile("mydatafile.txt")
baySearch <- BayesianSearch()
net <- baySearch$learn(ds)
```

Note that every variable in the dataset participates in the learning process. After learning the structure, each of the algorithms listed above automatically performs **parameter learning** using the Expectation-Maximization (EM) method, so the resulting network includes nodes with parameters estimated from the data.

The previous example used the default settings for Bayesian Search. To adjust the learning process, modify the control parameters of the learning object before calling its learn method. The example below demonstrates how to set the number of iterations and the maximum number of parents:

Python

```
baySearch = pysmile.learning.BayesianSearch()
baySearch.set_iteration_count(10)
baySearch.set_max_parents(4)
net = baySearch.learn(ds)
```

Java

```
BayesianSearch baySearch = new BayesianSearch();
baySearch.setIterationCount(10);
baySearch.setMaxParents(4);
Network net = baySearch.learn(ds);
```

C#

```
BayesianSearch baySearch = new BayesianSearch();
baySearch.IterationCount = 10;
baySearch.MaxParents = 4;
Network net = baySearch.Learn(ds);
```

R

```
baySearch <- BayesianSearch()
baySearch$setIterationCount(10)
baySearch$setMaxParents(4)
net <- baySearch$learn(ds)
```

For NaiveBayes and its semi-naive derivatives (TAN and ABN), a **class variable identifier** (string) must be specified by calling `set_class_variable_id` before invoking `learn`. This identifier must match one of the columns in the `DataSet` object passed to `learn`.

SMILE also provides the PC class, which implements the PC structure learning algorithm (its name is derived from the initials of its inventors). This algorithm uses a `DataSet` as its data source but learns a `Pattern` object instead of a `Network`. The `Pattern` represents a graph that may contain both directed and undirected edges and is not guaranteed to be acyclic.

Both the `BayesianSearch` and PC algorithms can incorporate **background knowledge** supplied by the caller. Background knowledge influences the learned structure by:

- **Forcing arcs** between specified variables.
- **Forbidding arcs** between specified variables.
- **Ordering specified variables** into temporal tiers, ensuring that no arcs are created from nodes in higher tiers to nodes in lower tiers.

To provide background knowledge, use the `BayesianSearch.set_bk_knowledge` or `PC.set_bk_knowledge` methods.

[Tutorial 10](#)¹⁶⁷ contains an example program demonstrating structure learning with Bayesian Search, Tree-Augmented Naive Bayes (TAN), and PC.

6.17.2 Learning network parameters

To **learn parameters** in an existing `Network` object, use the **Expectation-Maximization (EM)** algorithm implemented in the `EM` class. As with structure learning, the data is provided in a `DataSet` object. The network and dataset must be aligned so that the learning algorithm can determine the correspondence between dataset variables and network nodes.

If variable names and node identifiers are identical, the `DataSet.match_network` method can be used. This method creates an array of `DataMatch` objects that define the correspondence between each variable and its associated network node. `DataMatch` has `node` and `column` fields, representing the node handle and variable index, respectively. For each node/variable pair, one element of the array is required. When automatic matching is not possible, the user must create the `DataMatch` array manually and pass it to `EM.learn` to perform parameter learning.

Python

```
ds = pysmile.learning.DataSet()
net = pysmile.Network()
# load network and data here
matching = ds.match_network(net)
em = pysmile.learning.EM()
em.learn(ds, net, matching)
```

Java

```
DataSet ds = new DataSet();
Network net = new Network();
// load network and data here
DataMatch[] matching = ds.matchNetwork(net);
em = new EM();
em.learn(ds, net, matching);
```

C#

```
DataSet ds = new DataSet();
```

```

Network net = new Network();
// load network and data here
DataMatch[] matching = ds.MatchNetwork(net);
em = new EM();
em.Learn(ds, net, matching);

```

R

```

ds <- DataSet()
net <- Network()
# load network and data here
matching <- ds$matchNetwork(net)
em <- EM()
em$learn(ds, net, matching)

```

During EM parameter learning, nodes can be marked as *fixed*, meaning their conditional probability distributions remain unchanged throughout the learning process. Nodes not designated as fixed are subject to parameter updates, even if they are not matched to any variable in the dataset. To specify fixed nodes, pass an array of node handles or identifiers to the EM.Learn method.

The EM algorithm can start parameter learning from different initial conditions. The choice of **initialization method** affects convergence and, in some cases, the quality of the learned parameters.

1. **Uniformize parameters:** when this option is enabled, the algorithm initializes all network parameters with a uniform distribution. This approach effectively discards the existing parameters in the network. Use this option when you want to start learning from a neutral, non-informative prior. Call `EM.set_uniformize_parameters(true)` to enable uniform initialization.
2. **Randomize parameters (default):** random initialization assigns random values to all parameters. Although uniform initialization is common, it does not always yield better final parameter quality. The EM algorithm uses the initial parameter values only as a starting point in its search for the parameter set that maximizes the likelihood of the data given the model. Random initialization may be particularly useful when learning parameters in networks with latent variables, as it can help the algorithm avoid local maxima near uniform distributions. Call `EM.set_randomize_parameters(true)` to enable random initialization. The random seed can be specified with `EM.set_seed`. If not specified, or if zero is passed as a seed, the random generator will be seeded with system clock.
3. **Keep original parameters:** This option keeps the existing parameter values in the network and updates them based on the new data. It should be used when the new dataset is treated as an additional source of information rather than a complete replacement. In this case, the equivalent sample size (ESS) parameter becomes relevant. ESS represents the notional number of records that the current network parameters are based on. For networks whose parameters were previously learned from data, ESS should typically match the number of records in that dataset. The equivalent sample size defines the relative weight of the existing parameters compared to the new data. During EM learning, the algorithm combines prior information represented by the current parameters (weighted by ESS) with the likelihood derived from the dataset. A higher ESS makes the learned parameters change less in response to the new data, while a lower ESS allows the new data to have a stronger influence. Call `EM.set_uniformize_parameters(false)` and `EM.set_randomize_parameters(false)` to keep the original parameters. Set the equivalent sample size with `EM.set_eq_sample_size`.

The following methods of the EM class control parameter learning behavior, initialization options, and equivalent sample size.

Python

```

learn(data: DataSet, net: Network, matching: List[DataMatch], fixed_nodes: List[int] | List[str] | None = None
set_uniformize_parameters(value: bool)
get_uniformize_parameters() -> bool
set_randomize_parameters(value: bool)
get_randomize_parameters() -> bool

```

```

set_seed(seed: int)
get_seed() -> int
set_eq_sample_size(size: int)
get_eq_sample_size() -> int

```

Java

```

void learn(DataSet data, Network net, DataMatch[] matching, int[] fixedNodes);
void learn(DataSet data, Network net, DataMatch[] matching, String[] fixedNodes);
void learn(DataSet data, Network net, DataMatch[] matching);
void setUniformizeParameters(boolean value);
boolean getUniformizeParameters();
void setRandomizeParameters(boolean value);
boolean getRandomizeParameters();
void setSeed(int seed);
int getSeed();
void setEqSampleSize(int size);
int getEqSampleSize();

```

C#

```

void Learn(DataSet data, Network net, List<DataMatch> matching, int[] fixedNodes);
void Learn(DataSet data, Network net, List<DataMatch> matching, string[] fixedNodes1);
void Learn(DataSet data, Network net, List<DataMatch> matching);
bool UniformizeParameters { get; set; }
bool RandomizeParameters { get; set; }
int Seed { get; set; }
int EqSampleSize { get; set; }

```

R

```

learn(data, net, matching, fixedNodes = NULL)
set_uniformize_parameters(value)
uniformParams <- get_uniformize_parameters()
set_randomize_parameters(value)
randParams <- get_randomize_parameters()
set_seed(seed)
seed <- get_seed()
set_eq_sample_size(size)
size <- get_eq_sample_size()

```

The final section of [Tutorial 10](#)¹⁶⁷ demonstrates EM parameter learning applied to a network structure created by the PC algorithm.

6.17.3 Validation

To evaluate the predictive quality of a network, use the `Validator` class.

The `Validator` constructor requires references to both `DataSet` and `Network` objects. To correctly match network nodes to dataset variables, the constructor also requires an array of `DataMatch` objects, similar to the `EM.learn` method.

After constructing a validator object, specify which nodes in the network are considered class nodes by calling the `Validator.add_class_node` method. Validation requires at least one class node.

During validation, the variables matched to non-class nodes are used to set evidence for each record in the dataset. Posterior probabilities are then calculated, and for each class node, the outcome with the highest probability is selected as the predicted outcome. Predictions are compared with the corresponding outcomes in the dataset, and the results—including the number of matches and posterior probabilities—are used to calculate accuracy, the confusion matrix, ROC curves, and calibration curves.

Validation can be performed either without parameter learning using `Validator.test`, or with parameter learning using `Validator.k_fold` or `Validator.leave_one_out`. K-fold cross-validation divides the dataset into K parts of equal size, trains the network on K-1 parts, and tests it on the remaining part. This process is repeated K times, with each part used once for testing. Leave-one-out is an extreme case of K-fold, where K equals the number of records in the dataset.

The example below demonstrates 5-fold cross-validation using a single class node. Accuracy is obtained for the outcome with index zero, corresponding to the first outcome of the node.

Python

```
ds = pysmile.learning.DataSet()
net = pysmile.Network()
# load network and data here
matching = ds.match_network(net)
validator = pysmile.learning.Validator(ds, net, matching)
classNodehandle = net.getNode("someNodeId")
validator.addClassNode(classNodeHandle)
em = pysmile.learning.EM()
# optionally tweak EM options here
validator.k_fold(em, 5)
acc = validator.get_accuracy(classNodeHandle, 0)
```

Java

```
DataSet ds = new DataSet();
Network net = new Network();
// load network and data here
DataMatch[] matching = ds.matchNetwork();
Validator validator = new Validator(ds, net, matching);
int classNodehandle = net.getNode("someNodeId");
validator.addClassNode(classNodeHandle);
EM em = new EM();
// optionally tweak EM options here
validator.kFold(em, 5);
double acc = validator.getAccuracy(classNodeHandle, 0);
```

C#

```
DataSet ds = new DataSet();
Network net = new Network();
// load network and data here
DataMatch[] matching = ds.MatchNetwork();
Validator validator = new Validator(ds, net, matching);
int classNodehandle = net.GetNode("someNodeId");
validator.AddClassNode(classNodeHandle);
EM em = new EM();
// optionally tweak EM options here
validator.KFold(em, 5);
double acc = validator.GetAccuracy(classNodeHandle, 0);
```

R

```
ds <- DataSet()
net <- Network()
# load network and data here
matching <- ds$matchNetwork()
validator <- Validator(ds, net, matching)
classNodehandle <- net$getNode("someNodeId")
validator$addClassNode(classNodeHandle)
em <- EM()
# optionally tweak EM options here
validator$kFold(em, 5)
acc <- validator$getAccuracy(classNodeHandle, 0)
```

This page is intentionally left blank.

Tutorials

7 Tutorials

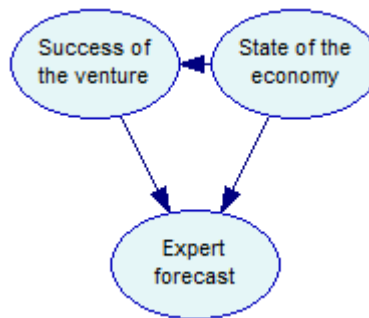
The complete, functionally equivalent source code for all tutorials is provided later in this chapter in Python, Java, C#, and R. Multi-line code snippets in the tutorial descriptions preceding the full source code are shown in Python, Java, C# and R. Inline examples within text paragraphs use Python syntax (for example, `Network.update_beliefs`) for brevity.

Tutorial sources can also be downloaded from the documentation site at <https://support.bayesfusion.com/docs>

7.1 Tutorial 1: Creating a Bayesian Network

Consider a slight twist on the problem described in the [Hello, SMILE Wrapper!](#)²² section of this manual.

The twist will include adding an additional variable *State of the economy* (with the identifier *Economy*) with three outcomes (*Up*, *Flat*, and *Down*) modeling the developments in the economy. These developments are relevant to the decision, as they are impacting expert predictions. When the economy is heading *Up*, our expert makes more optimistic predictions, when it is heading *Down*, the expert makes more pessimistic predictions for the same venture. This is reflected by a directed arc from the node *State of the economy* to the node *Expert forecast*. *State of the economy* also impacts the probability of venture being successful, which we model by adding another arc.



We will show how to create this model and how to save it to disk. In subsequent tutorials, we will show how to enter observations (evidence), how to perform inference, and how to retrieve the calculation results.

We start by declaring our network variable. The three nodes in the network are subsequently created by calling a helper method `createCptNode`.

Python:

```

net = pysmile.Network()
e = self.create_cpt_node(net,
    "Economy", "State of the economy",
    ["Up", "Flat", "Down"],
    160, 40)
s = self.create_cpt_node(net,
    "Success", "Success of the venture",
    ["Success", "Failure"],
    60, 40)
f = self.create_cpt_node(net,
    "Forecast", "Expert forecast",
    ["Good", "Moderate", "Poor"],
    110, 140)

```

Java:

```

Network net = new Network();

```

```

int e = createCptNode(net,
    "Economy", "State of the economy",
    new String[] { "Up", "Flat", "Down" },
    160, 40);
int s = createCptNode(net,
    "Success", "Success of the venture",
    new String[] { "Success", "Failure" },
    60, 40);
int f = createCptNode(net,
    "Forecast", "Expert forecast",
    new String[] { "Good", "Moderate", "Poor" },
    110, 140);

```

C#:

```

Network net = new Network();
int e = CreateCptNode(net,
    "Economy", "State of the economy",
    new String[] { "Up", "Flat", "Down" },
    160, 40);
int s = CreateCptNode(net,
    "Success", "Success of the venture",
    new String[] { "Success", "Failure" },
    60, 40);
int f = CreateCptNode(net,
    "Forecast", "Expert forecast",
    new String[] { "Good", "Moderate", "Poor" },
    110, 140);

```

R:

```

net <- Network()
e <- createCptNode(net, "Economy", "State of the economy",
    c("Up", "Flat", "Down"), 160L, 40L)

s <- createCptNode(net, "Success", "Success of the venture",
    c("Success", "Failure"), 60L, 40L)

f <- createCptNode(net, "Forecast", "Expert forecast",
    c("Good", "Moderate", "Poor"), 110L, 140L)

```

Before connecting the nodes with arcs, let's have a look at the createCptNode method.

Python:

```

def create_cpt_node(self, net, id, name, outcomes, x_pos, y_pos):
    handle = net.add_node(pysmile.NodeType.CPT, id)
    net.set_node_name(handle, name)
    net.set_node_position(handle, x_pos, y_pos, 85, 55)
    initial_outcome_count = net.get_outcome_count(handle)
    for i in range(0, initial_outcome_count):
        net.set_outcome_id(handle, i, outcomes[i])
    for i in range(initial_outcome_count, len(outcomes)):
        net.add_outcome(handle, outcomes[i])
    return handle

```

Java:

```

private static int createCptNode(
    Network net, String id, String name,
    String[] outcomes, int xPos, int yPos) {
    int handle = net.addNode(Network.NodeType.CPT, id);
    net.setNodeName(handle, name);
    net.setNodePosition(handle, xPos, yPos, 85, 55);
    int initialOutcomeCount = net.getOutcomeCount(handle);
    for (int i = 0; i < initialOutcomeCount; i++) {

```

```

        net.setOutcomeId(handle, i, outcomes[i]);
    }
    for (int i = initialOutcomeCount; i < outcomes.length; i++) {
        net.addOutcome(handle, outcomes[i]);
    }
    return handle;
}

```

C#:

```

private static int CreateCptNode(
    Network net, String id, String name,
    String[] outcomes, int xPos, int yPos)
{
    int handle = net.AddNode(Network.NodeType.Cpt, id);
    net.SetNodeName(handle, name);
    net.SetNodePosition(handle, xPos, yPos, 85, 55);
    int initialOutcomeCount = net.GetOutcomeCount(handle);
    for (int i = 0; i < initialOutcomeCount; i++)
    {
        net.SetOutcomeId(handle, i, outcomes[i]);
    }
    for (int i = initialOutcomeCount; i < outcomes.Length; i++)
    {
        net.AddOutcome(handle, outcomes[i]);
    }
    return handle;
}

```

R:

```

createCptNode = function(net, id, name, outcomes, xPos, yPos) {
    handle <- net$addNode(net$NodeType$CPT, id)

    net$setNodeName(handle, name)
    net$setNodePosition(handle, xPos, yPos, 85L, 55L)
    outcomesCount <- length(outcomes)
    initialOutcomeCount <- net$getOutcomeCount(handle)
    i <- 0
    for (outcome in outcomes[1:initialOutcomeCount]) {
        net$setOutcomeId(handle, i, outcome)
        i <- i + 1
    }

    if ((initialOutcomeCount + 1) <= outcomesCount) {
        for (outcome in outcomes[(initialOutcomeCount + 1):(outcomesCount)]) {
            net$addOutcome(handle, outcome)
        }
    }

    return(handle)
}

```

The function creates a CPT node with specified identifier, name, outcomes and position on the screen. CPT nodes are created with two outcomes named *State0* and *State1*. To change the number of node outcomes and rename them, we will use two loops - the first renames the default outcomes and the second adds new outcomes.

We can now add arcs linking the nodes. Note that we can use either node handles or node identifiers. In this simple program, we use both forms just to showcase this feature.

Python:

```

net.add_arc(e, s)
net.add_arc(s, f)
net.add_arc("Economy", "Forecast")

```

Java:

```
net.addArc(e, s);
net.addArc(s, f);
net.addArc("Economy", "Forecast");
```

C#:

```
net.AddArc(e, s);
net.AddArc(s, f);
net.AddArc("Economy", "Forecast");
```

R:

```
net$addArc(e, s)
net$addArc(s, f)
net$addArc("Economy", "Forecast")
```

Next step is to initialize the conditional probability tables of the nodes. See the [Multidimensional arrays](#) ³⁹ section in this manual for the description of the CPT memory layout. For each of three nodes we call the `Network.setNodeDefinition` method. Here's how we set the probabilities of the *Success* node - note that the comments for each probability show the combination of outcomes it is defined for.

Python:

```
successDef = [
    0.3, # P(Success=S|Economy=U)
    0.7, # P(Success=F|Economy=U)
    0.2, # P(Success=S|Economy=F)
    0.8, # P(Success=F|Economy=F)
    0.1, # P(Success=S|Economy=D)
    0.9 # P(Success=F|Economy=D)
]
net.set_node_definition(s, successDef)
```

Java:

```
double[] successDef = new double[] {
    0.3, // P(Success=S|Economy=U)
    0.7, // P(Success=F|Economy=U)
    0.2, // P(Success=S|Economy=F)
    0.8, // P(Success=F|Economy=F)
    0.1, // P(Success=S|Economy=D)
    0.9 // P(Success=F|Economy=D)
};
net.setNodeDefinition(s, successDef);
```

C#:

```
double[] successDef = new double[]
{
    0.3, // P(Success=S|Economy=U)
    0.7, // P(Success=F|Economy=U)
    0.2, // P(Success=S|Economy=F)
    0.8, // P(Success=F|Economy=F)
    0.1, // P(Success=S|Economy=D)
    0.9 // P(Success=F|Economy=D)
};
net.SetNodeDefinition(s, successDef);
```

R:

```
successDef <- c(0.3, # P(Success=S|Economy=U)
               0.7, # P(Success=F|Economy=U)
               0.2, # P(Success=S|Economy=F)
               0.8, # P(Success=F|Economy=F)
               0.1, # P(Success=S|Economy=D)
               0.9 # P(Success=F|Economy=D)
               )
net$setNodeDefinition(s, successDef)
```

With CPTs initialized our network is complete. We write its contents to the tutorial1.xdsl file. Tutorial 2 will load this file and perform the inference. The split between tutorials is artificial, your program can use networks right after its creation without the need to write/read from the file system.

Python:

```
net.write_file("tutorial1.xdsl");
```

Java:

```
net.writeFile("tutorial1.xdsl");
```

C#:

```
net.WriteFile("tutorial1.xdsl");
```

R:

```
net$writeFile("tutorial1.xdsl")
```

7.1.1 Tutorial1.py

```
import pysmile

# Tutorial1 creates a simple network with three nodes,
# then writes its content as XDSL file to disk.

class Tutorial1:
    def __init__(self):
        print("Starting Tutorial1...")
        net = pysmile.Network()

        e = self.create_cpt_node(net,
                                "Economy", "State of the economy",
                                ["Up", "Flat", "Down"],
                                160, 40)

        s = self.create_cpt_node(net,
                                "Success", "Success of the venture",
                                ["Success", "Failure"],
                                60, 40)

        f = self.create_cpt_node(net,
                                "Forecast", "Expert forecast",
                                ["Good", "Moderate", "Poor"],
                                110, 140)

        net.add_arc(e, s)
        net.add_arc(s, f)

        # we can also use node identifiers when creating arcs
        net.add_arc("Economy", "Forecast");

        economyDef = [
            0.2, # P(Economy=U)
            0.7, # P(Economy=F)
            0.1 # P(Economy=D)
        ]
        net.set_node_definition(e, economyDef)

        successDef = [
            0.3, # P(Success=S|Economy=U)
            0.7, # P(Success=F|Economy=U)
            0.2, # P(Success=S|Economy=F)
            0.8, # P(Success=F|Economy=F)
            0.1, # P(Success=S|Economy=D)
```

```

    0.9 # P(Success=F|Economy=D)
]
net.set_node_definition(s, successDef)

forecastDef = [
    0.70, # P(Forecast=G|Success=S,Economy=U)
    0.29, # P(Forecast=M|Success=S,Economy=U)
    0.01, # P(Forecast=P|Success=S,Economy=U)

    0.65, # P(Forecast=G|Success=S,Economy=F)
    0.30, # P(Forecast=M|Success=S,Economy=F)
    0.05, # P(Forecast=P|Success=S,Economy=F)

    0.60, # P(Forecast=G|Success=S,Economy=D)
    0.30, # P(Forecast=M|Success=S,Economy=D)
    0.10, # P(Forecast=P|Success=S,Economy=D)

    0.15, # P(Forecast=G|Success=F,Economy=U)
    0.30, # P(Forecast=M|Success=F,Economy=U)
    0.55, # P(Forecast=P|Success=F,Economy=U)

    0.10, # P(Forecast=G|Success=F,Economy=F)
    0.30, # P(Forecast=M|Success=F,Economy=F)
    0.60, # P(Forecast=P|Success=F,Economy=F)

    0.05, # P(Forecast=G|Success=F,Economy=D)
    0.25, # P(Forecast=M|Success=F,Economy=D)
    0.70 # P(Forecast=P|Success=F,Economy=D)
]
net.set_node_definition(f, forecastDef)

net.write_file("tutorial1.xdsl")

print("Tutorial1 complete: Network written to tutorial1.xdsl")

def create_cpt_node(self, net, id, name, outcomes, x_pos, y_pos):
    handle = net.add_node(pysmile.NodeType.CPT, id)

    net.set_node_name(handle, name)
    net.set_node_position(handle, x_pos, y_pos, 85, 55)

    initial_outcome_count = net.get_outcome_count(handle)

    for i in range(0, initial_outcome_count):
        net.set_outcome_id(handle, i, outcomes[i])

    for i in range(initial_outcome_count, len(outcomes)):
        net.add_outcome(handle, outcomes[i])

    return handle

```

7.1.2 Tutorial1.java

```

package tutorials;

import smile.*;

// Tutorial1 creates a simple network with three nodes,
// then writes its content as XDSL file to disk.

```

```

public class Tutorial1 {
    public static void run() {
        System.out.println("Starting Tutorial1...");
        Network net = new Network();

        int e = createCptNode(net,
            "Economy", "State of the economy",
            new String[] {"Up", "Flat", "Down"},
            160, 40);

        int s = createCptNode(net,
            "Success", "Success of the venture",
            new String[] {"Success", "Failure"},
            60, 40);

        int f = createCptNode(net,
            "Forecast", "Expert forecast",
            new String[] {"Good", "Moderate", "Poor"},
            110, 140);

        net.addArc(e, s);
        net.addArc(s, f);

        // we can also use node identifiers when creating arcs
        net.addArc("Economy", "Forecast");

        double[] economyDef = {
            0.2, // P(Economy=U)
            0.7, // P(Economy=F)
            0.1 // P(Economy=D)
        };
        net.setNodeDefinition(e, economyDef);

        double[] successDef = new double[] {
            0.3, // P(Success=S|Economy=U)
            0.7, // P(Success=F|Economy=U)
            0.2, // P(Success=S|Economy=F)
            0.8, // P(Success=F|Economy=F)
            0.1, // P(Success=S|Economy=D)
            0.9 // P(Success=F|Economy=D)
        };
        net.setNodeDefinition(s, successDef);

        double[] forecastDef = new double[] {
            0.70, // P(Forecast=G|Success=S,Economy=U)
            0.29, // P(Forecast=M|Success=S,Economy=U)
            0.01, // P(Forecast=P|Success=S,Economy=U)

            0.65, // P(Forecast=G|Success=S,Economy=F)
            0.30, // P(Forecast=M|Success=S,Economy=F)
            0.05, // P(Forecast=P|Success=S,Economy=F)

            0.60, // P(Forecast=G|Success=S,Economy=D)
            0.30, // P(Forecast=M|Success=S,Economy=D)
            0.10, // P(Forecast=P|Success=S,Economy=D)

            0.15, // P(Forecast=G|Success=F,Economy=U)
            0.30, // P(Forecast=M|Success=F,Economy=U)
            0.55, // P(Forecast=P|Success=F,Economy=U)

            0.10, // P(Forecast=G|Success=F,Economy=F)
        };
    }
}

```

```

        0.30, // P(Forecast=M|Success=F,Economy=F)
        0.60, // P(Forecast=P|Success=F,Economy=F)

        0.05, // P(Forecast=G|Success=F,Economy=D)
        0.25, // P(Forecast=G|Success=F,Economy=D)
        0.70  // P(Forecast=G|Success=F,Economy=D)
    };
    net.setNodeDefinition(f, forecastDef);

    net.writeFile("tutorial1.xdsl");

    System.out.println("Tutorial1 complete: Network written to tutorial1.xdsl");
}

private static int createCptNode(
    Network net, String id, String name,
    String[] outcomes, int xPos, int yPos) {
    int handle = net.addNode(Network.NodeType.CPT, id);

    net.setNodeName(handle, name);
    net.setNodePosition(handle, xPos, yPos, 85, 55);

    int initialOutcomeCount = net.getOutcomeCount(handle);
    for (int i = 0; i < initialOutcomeCount; i++) {
        net.setOutcomeId(handle, i, outcomes[i]);
    }

    for (int i = initialOutcomeCount; i < outcomes.length; i++) {
        net.addOutcome(handle, outcomes[i]);
    }

    return handle;
}
}

```

7.1.3 Tutorial1.cs

```

using System;
using Smile;

// Tutorial1 creates a simple network with three nodes,
// then writes its content as XDSL file to disk.

namespace SmileNetTutorial
{
    class Tutorial1
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial1...");
            Network net = new Network();

            int e = CreateCptNode(net,
                "Economy", "State of the economy",
                new String[] { "Up", "Flat", "Down" },
                160, 40);

            int s = CreateCptNode(net,
                "Success", "Success of the venture",
                new String[] { "Success", "Failure" },

```

```

        60, 40);

int f = CreateCptNode(net,
    "Forecast", "Expert forecast",
    new String[] { "Good", "Moderate", "Poor" },
    110, 140);

net.AddArc(e, s);
net.AddArc(s, f);

// we can also use node identifiers when creating arcs
net.AddArc("Economy", "Forecast");

double[] economyDef =
{
    0.2, // P(Economy=U)
    0.7, // P(Economy=F)
    0.1  // P(Economy=D)
};
net.SetNodeDefinition(e, economyDef);

double[] successDef = new double[]
{
    0.3, // P(Success=S|Economy=U)
    0.7, // P(Success=F|Economy=U)
    0.2, // P(Success=S|Economy=F)
    0.8, // P(Success=F|Economy=F)
    0.1, // P(Success=S|Economy=D)
    0.9  // P(Success=F|Economy=D)
};
net.SetNodeDefinition(s, successDef);

double[] forecastDef = new double[]
{
    0.70, // P(Forecast=G|Success=S,Economy=U)
    0.29, // P(Forecast=M|Success=S,Economy=U)
    0.01, // P(Forecast=P|Success=S,Economy=U)

    0.65, // P(Forecast=G|Success=S,Economy=F)
    0.30, // P(Forecast=M|Success=S,Economy=F)
    0.05, // P(Forecast=P|Success=S,Economy=F)

    0.60, // P(Forecast=G|Success=S,Economy=D)
    0.30, // P(Forecast=M|Success=S,Economy=D)
    0.10, // P(Forecast=P|Success=S,Economy=D)

    0.15, // P(Forecast=G|Success=F,Economy=U)
    0.30, // P(Forecast=M|Success=F,Economy=U)
    0.55, // P(Forecast=P|Success=F,Economy=U)

    0.10, // P(Forecast=G|Success=F,Economy=F)
    0.30, // P(Forecast=M|Success=F,Economy=F)
    0.60, // P(Forecast=P|Success=F,Economy=F)

    0.05, // P(Forecast=G|Success=F,Economy=D)
    0.25, // P(Forecast=M|Success=F,Economy=D)
    0.70  // P(Forecast=P|Success=F,Economy=D)
};
net.SetNodeDefinition(f, forecastDef);

net.WriteFile("tutorial11.xdsl");

```

```

        Console.WriteLine("Tutorial1 complete: Network written to tutorial1.xdsl");
    }

    private static int CreateCptNode(
        Network net, String id, String name,
        String[] outcomes, int xPos, int yPos)
    {
        int handle = net.AddNode(Network.NodeType.Cpt, id);

        net.SetNodeName(handle, name);
        net.SetNodePosition(handle, xPos, yPos, 85, 55);

        int initialOutcomeCount = net.GetOutcomeCount(handle);
        for (int i = 0; i < initialOutcomeCount; i++)
        {
            net.SetOutcomeId(handle, i, outcomes[i]);
        }

        for (int i = initialOutcomeCount; i < outcomes.Length; i++)
        {
            net.AddOutcome(handle, outcomes[i]);
        }

        return handle;
    }
}

```

7.1.4 Tutorial1.R

```

library(rSMILE)
source("License.R")

createCptNode = function(net, id, name, outcomes, xPos, yPos) {
    handle <- net$addNode(net$NodeType$CPT, id)

    net$setNodeName(handle, name)
    net$setNodePosition(handle, xPos, yPos, 85L, 55L)
    outcomesCount <- length(outcomes)
    initialOutcomeCount <- net$getOutcomeCount(handle)
    i <- 0
    for (outcome in outcomes[1:initialOutcomeCount]) {
        net$setOutcomeId(handle, i, outcome)
        i <- i + 1
    }

    if ((initialOutcomeCount + 1) <= outcomesCount) {
        for (outcome in outcomes[(initialOutcomeCount + 1):(outcomesCount)]) {
            net$addOutcome(handle, outcome)
        }
    }

    return(handle)
}

cat("Starting Tutorial1...\n")

net <- Network()
e <- createCptNode(net, "Economy", "State of the economy",

```

```

c("Up", "Flat", "Down"), 160L, 40L)

s <- createCptNode(net, "Success", "Success of the venture",
  c("Success", "Failure"), 60L, 40L)

f <- createCptNode(net, "Forecast", "Expert forecast",
  c("Good", "Moderate", "Poor"), 110L, 140L)

net$addArc(e, s)
net$addArc(s, f)

net$addArc("Economy", "Forecast")

economyDef <- c(0.2, # P(Economy=U)
  0.7, # P(Economy=F)
  0.1 # P(Economy=D)
)
net$setNodeDefinition(e, economyDef)

successDef <- c(0.3, # P(Success=S|Economy=U)
  0.7, # P(Success=F|Economy=U)
  0.2, # P(Success=S|Economy=F)
  0.8, # P(Success=F|Economy=F)
  0.1, # P(Success=S|Economy=D)
  0.9 # P(Success=F|Economy=D)
)
net$setNodeDefinition(s, successDef)

forecastDef <- c(0.70, # P(Forecast=G|Success=S,Economy=U)
  0.29, # P(Forecast=M|Success=S,Economy=U)
  0.01, # P(Forecast=P|Success=S,Economy=U)

  0.65, # P(Forecast=G|Success=S,Economy=F)
  0.30, # P(Forecast=M|Success=S,Economy=F)
  0.05, # P(Forecast=P|Success=S,Economy=F)

  0.60, # P(Forecast=G|Success=S,Economy=D)
  0.30, # P(Forecast=M|Success=S,Economy=D)
  0.10, # P(Forecast=P|Success=S,Economy=D)

  0.15, # P(Forecast=G|Success=F,Economy=U)
  0.30, # P(Forecast=M|Success=F,Economy=U)
  0.55, # P(Forecast=P|Success=F,Economy=U)

  0.10, # P(Forecast=G|Success=F,Economy=F)
  0.30, # P(Forecast=M|Success=F,Economy=F)
  0.60, # P(Forecast=P|Success=F,Economy=F)

  0.05, # P(Forecast=G|Success=F,Economy=D)
  0.25, # P(Forecast=M|Success=F,Economy=D)
  0.70 # P(Forecast=P|Success=F,Economy=D)
)
net$setNodeDefinition(f, forecastDef)

net$writeFile("tutorial1.xdsl")

cat("Tutorial1 complete: Network written to tutorial1.xdsl\n")

```

7.2 Tutorial 2: Inference with a Bayesian Network

Tutorial 2 begins with the model we have previously created. We will perform multiple calls to Bayesian inference algorithm through the `Network.updateBeliefs`, starting with network without any evidence. After each `updateBeliefs` call the posterior probabilities of nodes will be displayed.

The model is loaded with the `Network.readFile`.

Python:

```
net.read_file("tutorial1.xdsl");
```

Java:

```
net.readFile("tutorial1.xdsl");
```

C#:

```
net.ReadFile("tutorial1.xdsl");
```

R:

```
net$readFile("tutorial1.xdsl")
```

We update the probabilities and proceed to display them using the helper method `printAllPosteriors` defined in this tutorial.

Python:

```
print("Posteriors with no evidence set:")
net.update_beliefs()
self.print_all_posteriors(net)
```

Java:

```
printf("Posteriors with no evidence set:\n");
net.updateBeliefs();
printAllPosteriors(net);
```

C#:

```
Console.WriteLine("Posteriors with no evidence set:");
net.UpdateBeliefs();
PrintAllPosteriors(net);
```

R:

```
cat("Posteriors with no evidence set:\n")
net$updateBeliefs()
printAllPosteriors(net)
```

`printAllPosteriors` displays posterior probabilities calculated by `updateBeliefs` for each node. To iterate over the nodes, `Network.getFirstNode` and `getNextNode` are used. In the body of the loop we call another helper function, `printPosteriors`.

Python:

```
handle = net.get_first_node()
while (handle >= 0):
    self.print_posteriors(net, handle)
    handle = net.get_next_node(handle)
```

Java:

```
for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h)) {
    printPosteriors(net, h);
}
```

C#:

```
for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
{
    PrintPosteriors(net, h);
}
```

R:

```
nodes <- net$getAllNodes()
```

```

for (h in nodes) {
    printPosteriors(net, h)
}

```

`printPosteriors` checks if node has evidence set by calling `Network.isEvidence`; if this is the case the name of the evidence state is displayed. Please note that for demonstration purposes this tutorial calls `Network.getEvidence` to obtain an integer representing the index of the evidence state, then converts the integer index to outcome id with `Network.getOutcomeId`. The `Network.getEvidenceId` method combines these two actions into one call.

If node does not have any evidence set, `printPosteriors` iterates over all states and displays the posterior probability of each.

Back in the main function of the tutorial, we repeatedly call `changeEvidenceAndUpdate` helper function to change evidence, update network and display posteriors. Note that we need to call different methods of the `Network` class to set evidence to the specified outcome (`setEvidence`) and remove the evidence (`clearEvidence`).

7.2.1 Tutorial2.py

```

import pysmile

# Tutorial2 loads the XDSL file created by Tutorial1,
# then performs the series of inference calls,
# changing evidence each time.

class Tutorial2:
    def __init__(self):
        print("Starting Tutorial2...")
        net = pysmile.Network()

        # load the network created by Tutorial1
        net.read_file("tutorial1.xdsl")

        print("Posteriors with no evidence set:")
        net.update_beliefs()
        self.print_all_posteriors(net)

        print("Setting Forecast=Good.")
        self.change_evidence_and_update(net, "Forecast", "Good")

        print("Adding Economy=Up.")
        self.change_evidence_and_update(net, "Economy", "Up")

        print("Changing Forecast to Poor, keeping Economy=Up.")
        self.change_evidence_and_update(net, "Forecast", "Poor")

        print("Removing evidence from Economy, keeping Forecast=Poor.")
        self.change_evidence_and_update(net, "Economy", None)

        print("Tutorial2 complete.")

    def print_posteriors(self, net, node_handle):
        node_id = net.get_node_id(node_handle)
        if net.is_evidence(node_handle):
            print(f"{node_id} has evidence set ({net.get_evidence_id(node_handle)})")
        else :
            posteriors = net.get_node_value(node_handle)
            for i in range(0, len(posterior)):
                print(f"P({node_id}={net.get_outcome_id(node_handle, i)})={posteriors[i]}")

```

```

def print_all_posteriors(self, net):
    for handle in net.get_all_nodes():
        self.print_posteriors(net, handle)

def change_evidence_and_update(self, net, node_id, outcome_id):
    if outcome_id is not None:
        net.set_evidence(node_id, outcome_id)
    else:
        net.clear_evidence(node_id)
    net.update_beliefs()
    self.print_all_posteriors(net)
    print()

```

7.2.2 Tutorial2.java

```

package tutorials;

import smile.*;

// Tutorial2 loads the XDSL file created by Tutorial1,
// then performs the series of inference calls,
// changing evidence each time.

public class Tutorial2 {
    public static void run() {
        System.out.println("Starting Tutorial2...");
        Network net = new Network();

        // load the network created by Tutorial1
        net.readFile("tutorial1.xdsl");

        System.out.println("Posteriors with no evidence set:");
        net.updateBeliefs();
        printAllPosteriors(net);

        System.out.println("Setting Forecast=Good.");
        changeEvidenceAndUpdate(net, "Forecast", "Good");

        System.out.println("Adding Economy=Up.");
        changeEvidenceAndUpdate(net, "Economy", "Up");

        System.out.println("Changing Forecast to Poor, keeping Economy=Up.");
        changeEvidenceAndUpdate(net, "Forecast", "Poor");

        System.out.println("Removing evidence from Economy, keeping Forecast=Poor.");
        changeEvidenceAndUpdate(net, "Economy", null);

        System.out.println("Tutorial2 complete.");
    }

    private static void printPosteriors(Network net, int nodeHandle) {
        String nodeId = net.getNodeId(nodeHandle);
        if (net.isEvidence(nodeHandle)) {
            System.out.printf("%s has evidence set (%s)\n",
                nodeId,
                net.getOutcomeId(nodeHandle, net.getEvidence(nodeHandle)));
        } else {
            double[] posteriors = net.getNodeValue(nodeHandle);
            for (int i = 0; i < posteriors.length; i++) {

```

```

        System.out.printf("P(%s=%s)=%f\n",
            nodeId,
            net.getOutcomeId(nodeHandle, i),
            posteriors[i]);
    }
}

private static void printAllPosteriors(Network net) {
    for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h)) {
        printPosteriors(net, h);
    }
    System.out.println();
}

private static void changeEvidenceAndUpdate(
    Network net, String nodeId, String outcomeId) {
    if (outcomeId != null) {
        net.setEvidence(nodeId, outcomeId);
    } else {
        net.clearEvidence(nodeId);
    }

    net.updateBeliefs();
    printAllPosteriors(net);
}
}

```

7.2.3 Tutorial2.cs

```

using System;
using Smile;

// Tutorial2 loads the XDSL file created by Tutorial1,
// then performs the series of inference calls,
// changing evidence each time.

namespace SmileNetTutorial
{
    class Tutorial2
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial2...");
            Network net = new Network();

            // load the network created by Tutorial1
            net.ReadFile("tutorial1.xdsl");

            Console.WriteLine("Posteriors with no evidence set:");
            net.UpdateBeliefs();
            PrintAllPosteriors(net);

            Console.WriteLine("Setting Forecast=Good.");
            ChangeEvidenceAndUpdate(net, "Forecast", "Good");

            Console.WriteLine("Adding Economy=Up.");
            ChangeEvidenceAndUpdate(net, "Economy", "Up");

            Console.WriteLine("Changing Forecast to Poor, keeping Economy=Up.");
        }
    }
}

```

```

        ChangeEvidenceAndUpdate(net, "Forecast", "Poor");

        Console.WriteLine("Removing evidence from Economy, keeping Forecast=Poor.");
        ChangeEvidenceAndUpdate(net, "Economy", null);

        Console.WriteLine("Tutorial2 complete.");
    }

    private static void PrintPosteriors(Network net, int nodeHandle)
    {
        String nodeId = net.GetNodeId(nodeHandle);
        if (net.IsEvidence(nodeHandle))
        {
            Console.WriteLine("{0} has evidence set ({1})",
                nodeId,
                net.GetOutcomeId(nodeHandle, net.GetEvidence(nodeHandle)));
        }
        else
        {
            double[] posteriors = net.GetNodeValue(nodeHandle);
            for (int i = 0; i < posteriors.Length; i++)
            {
                Console.WriteLine("P({0}={1})={2}",
                    nodeId, net.GetOutcomeId(nodeHandle, i), posteriors[i]);
            }
        }
    }

    private static void PrintAllPosteriors(Network net)
    {
        for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
        {
            PrintPosteriors(net, h);
        }
        Console.WriteLine();
    }

    private static void ChangeEvidenceAndUpdate(
        Network net, String nodeId, String outcomeId)
    {
        if (outcomeId != null)
        {
            net.SetEvidence(nodeId, outcomeId);
        }
        else
        {
            net.ClearEvidence(nodeId);
        }

        net.UpdateBeliefs();
        PrintAllPosteriors(net);
    }
}

```

7.2.4 Tutorial2.R

```

library(rSMILE)
source("License.R")

```

```

printPosteriors = function(net, nodeHandle) {
  nodeId <- net$getNodeId(nodeHandle)
  if (net$isEvidence(nodeHandle)) {
    cat(sprintf("%s has evidence set (%s)\n",
              nodeId,
              net$getOutcomeId(nodeHandle, net$getEvidence(nodeHandle))))

  } else {
    posteriors <- net$getNodeValue(nodeHandle)
    for (i in 0:(length(posteriors)-1)) {
      cat(sprintf("P(%s=%s)=%f\n",
                nodeId, net$getOutcomeId(nodeHandle, i), posteriors[i+1]))
    }
  }
}

printAllPosteriors = function(net) {
  nodes <- net$getAllNodes()
  for (h in nodes) {
    printPosteriors(net, h)
  }
  cat("\n")
}

changeEvidenceAndUpdate = function(net, nodeId, outcomeId) {
  if (is.null(outcomeId)) {
    net$clearEvidence(nodeId)
  } else {
    net$setEvidence(nodeId, outcomeId)
  }
  net$updateBeliefs()
  printAllPosteriors(net)
}

cat("Starting Tutorial2...\n")
net <- Network()

# load the network created by Tutorial1
net$readFile("tutorial1.xdsl")

cat("Posteriors with no evidence set:\n")
net$updateBeliefs()
printAllPosteriors(net)

cat("Setting Forecast=Good.\n");
changeEvidenceAndUpdate(net, "Forecast", "Good");

cat("Adding Economy=Up.\n");
changeEvidenceAndUpdate(net, "Economy", "Up");

cat("Changing Forecast to Poor, keeping Economy=Up.\n");
changeEvidenceAndUpdate(net, "Forecast", "Poor");

cat("Removing evidence from Economy, keeping Forecast=Poor.\n");
changeEvidenceAndUpdate(net, "Economy", NULL);

cat("Tutorial2 complete.\n");

```

7.3 Tutorial 3: Exploring the contents of a model

This tutorial will not perform any calculations. Instead, we will display the information about the network structure (nodes and arcs) and parameters (in this case, conditional probability tables).

We load the network created by Tutorial1 and for each node invoke the locally defined helper method `printNodeInfo`. This is where real work is done. The first displayed node attributes are identifier and name.

Python:

```
print("Node id/name: " + net.get_node_id(node_handle) + "/" +
      net.get_node_name(node_handle))
```

Java:

```
System.out.printf("Node id/name: %s/%s\n",
    net.getNodeId(nodeHandle),
    net.getNodeName(nodeHandle));
```

C#:

```
Console.WriteLine("Node id/name: {0}/{1}",
    net.GetNodeId(nodeHandle),
    net.GetNodeName(nodeHandle));
```

R:

```
cat(sprintf("Node id/name: %s/%s\n",
    net$getNodeId(nodeHandle),
    net$getNodeName(nodeHandle)))
```

The identifiers of node's outcomes are next.

Python:

```
print(" Outcomes: " + " ".join(net.get_outcome_ids(node_handle)))
```

Java:

```
System.out.print(" Outcomes:");
for (String outcomeId: net.getOutcomeIds(nodeHandle)) {
    System.out.print(" " + outcomeId);
}
```

C#:

```
foreach (String outcomeId in net.GetOutcomeIds(nodeHandle))
{
    Console.Write(" " + outcomeId);
}
```

R:

```
cat(" Outcomes: ")
cat(paste(net$getOutcomeIds(nodeHandle), collapse=" "))
```

The parents of the node follow.

Python:

```
parent_ids = net.get_parent_ids(node_handle)
if len(parent_ids) > 0:
    print(" Parents: " + " ".join(parent_ids))
```

Java:

```
String[] parentIds = net.getParentIds(nodeHandle);
if (parentIds.length > 0) {
    System.out.print(" Parents:");
    for (String parentId: parentIds) {
        System.out.print(" " + parentId);
    }
    System.out.println();
}
```

C#:

```
String[] parentIds = net.GetParentIds(nodeHandle);
if (parentIds.Length > 0)
{
    Console.Write("  Parents:");
    foreach (String parentId in parentIds)
    {
        Console.Write(" " + parentId);
    }
    Console.WriteLine();
}
```

R:

```
parentIds <- net$getParentIds(nodeHandle)
if (length(parentIds) > 0) {
  cat("  Parents: ")
  cat(paste(parentIds, collapse=" "))
}
```

We proceed to display information about node's children. The code fragment is virtually identical to the iteration over parents. Note that while tutorial uses `Network.getParentIds` and `getChildIds` to obtain identifiers of related nodes, we could alternatively use the `Network.getParents` and `getChildren`, which return the node handles (integer numbers) instead of identifiers (strings).

Finally, the node probabilities are displayed by the `printCptMatrix` helper. We retrieve the single-dimensional array containing the probabilities with `Network.getNodeDefinition` and iterate over its elements, translating each index into a multi-dimensional coordinates of the elements within node's CPT. This conversion is performed in the helper method `indexToCoords`, which requires the information about the outcome counts of the node and its parents in its `dimSizes` input parameter. The first part of `printCptMatrix` obtains the CPT and initializes `dimSizes`

Python:

```
cpt = net.get_node_definition(node_handle)
parents = net.get_parents(node_handle)
dim_count = 1 + len(parents)
dim_sizes = [0] * dim_count
for i in range(0, dim_count - 1):
    dim_sizes[i] = net.get_outcome_count(parents[i])
dim_sizes[len(dim_sizes) - 1] = net.get_outcome_count(node_handle)
```

Java:

```
double[] cpt = net.getNodeDefinition(nodeHandle);
int[] parents = net.getParents(nodeHandle);
int dimCount = 1 + parents.length;
int[] dimSizes = new int[dimCount];
for (int i = 0; i < dimCount - 1; i++) {
    dimSizes[i] = net.getOutcomeCount(parents[i]);
}
dimSizes[dimSizes.length - 1] = net.getOutcomeCount(nodeHandle);
```

C#:

```
double[] cpt = net.GetNodeDefinition(nodeHandle);
int[] parents = net.GetParents(nodeHandle);
int dimCount = 1 + parents.Length;
int[] dimSizes = new int[dimCount];
for (int i = 0; i < dimCount - 1; i++)
{
    dimSizes[i] = net.GetOutcomeCount(parents[i]);
}
dimSizes[dimSizes.Length - 1] = net.GetOutcomeCount(nodeHandle);
```

R:

```
cpt <- net$getNodeDefinition(nodeHandle)
parents <- net$getParents(nodeHandle)
```

```
dimCount <- 1 + length(parents)

dimSizes <- sapply(parents, function(x) net$getOutcomeCount(x))
dimSizes[length(dimSizes)+1] <- net$getOutcomeCount(nodeHandle)
```

The main loop in the `printCptMatrix` uses performs the iteration over the elements of the CPT. The `coords` integer array is filled by `indexToCoords` and used to print the textual information about the outcome of the node and parents for each element. Note that the node outcome is the rightmost entry in the `coords`. The parents' outcome indexes start from the left at index 0 in the `coords`. Part of the Tutorial3 output for the node *Forecast* is show below. All lines starting with “P”(were printed by `printCptMatrix`.

```
Node: Expert forecast
Outcomes: Good Moderate Poor
Parents: Success Economy
Definition type: CPT
P(Good | Success=Success,Economy=Up)=0.7
P(Moderate | Success=Success,Economy=Up)=0.29
P(Poor | Success=Success,Economy=Up)=0.01
P(Good | Success=Success,Economy=Flat)=0.65
P(Moderate | Success=Success,Economy=Flat)=0.3
P(Poor | Success=Success,Economy=Flat)=0.05
P(Good | Success=Success,Economy=Down)=0.6
P(Moderate | Success=Success,Economy=Down)=0.3
P(Poor | Success=Success,Economy=Down)=0.1
P(Good | Success=Failure,Economy=Up)=0.15
P(Moderate | Success=Failure,Economy=Up)=0.3
P(Poor | Success=Failure,Economy=Up)=0.55
P(Good | Success=Failure,Economy=Flat)=0.1
P(Moderate | Success=Failure,Economy=Flat)=0.3
P(Poor | Success=Failure,Economy=Flat)=0.6
P(Good | Success=Failure,Economy=Down)=0.05
P(Moderate | Success=Failure,Economy=Down)=0.25
P(Poor | Success=Failure,Economy=Down)=0.7
```

7.3.1 Tutorial3.py

```
import pysmile

# Tutorial3 loads the XDSL file and prints the information
# about the structure (nodes and arcs) and the parameters
# (conditional probabilities of the nodes) of the network.

class Tutorial3:
    def __init__(self):
        print("Starting Tutorial3...")
        net = pysmile.Network()

        # load the network created by Tutorial1
        net.read_file("tutorial1.xdsl")
        for h in net.get_all_nodes():
            self.print_node_info(net, h)
        print("Tutorial3 complete.")

    def print_node_info(self, net, node_handle):
        print(f"Node id/name: {net.get_node_id(node_handle)}/{net.get_node_name(node_handle)}")
        print(f" Outcomes: {' '.join(net.get_outcome_ids(node_handle))}")

        parent_ids = net.get_parent_ids(node_handle)
        if len(parent_ids) > 0:
```

```

        print(f"  Parents: {' '.join(parent_ids)}")
        child_ids = net.get_child_ids(node_handle)
        if len(child_ids) > 0:
            print(f"  Children: {' '.join(child_ids)}")

        self.print_cpt_matrix(net, node_handle)

def print_cpt_matrix(self, net, node_handle):
    cpt = net.get_node_definition(node_handle)
    parents = net.get_parents(node_handle)
    dim_count = 1 + len(parents)

    dim_sizes = [0] * dim_count
    for i in range(0, dim_count - 1):
        dim_sizes[i] = net.get_outcome_count(parents[i])
    dim_sizes[len(dim_sizes) - 1] = net.get_outcome_count(node_handle)

    coords = [0] * dim_count
    for elem_idx in range(0, len(cpt)):
        self.index_to_coords(elem_idx, dim_sizes, coords)

        outcome = net.get_outcome_id(node_handle, coords[dim_count - 1])
        print(f"      P({outcome}), end="")

        if dim_count > 1:
            print(" | ", end="")
            for parent_idx in range(0, len(parents)):
                if parent_idx > 0:
                    print(", ", end="")
                parent_handle = parents[parent_idx]
                print(f"{net.get_node_id(parent_handle)}="
                    + f"{net.get_outcome_id(parent_handle, coords[parent_idx])}", end="")

        prob = cpt[elem_idx]
        print(f")={prob}")

def index_to_coords(self, index, dim_sizes, coords):
    prod = 1
    for i in range(len(dim_sizes) - 1, -1, -1):
        coords[i] = int(index / prod) % dim_sizes[i]
        prod *= dim_sizes[i]

```

7.3.2 Tutorial3.java

```

package tutorials;

import smile.*;

// Tutorial3 loads the XDSL file and prints the information
// about the structure (nodes and arcs) and the parameters
// (conditional probabilities of the nodes) of the network.

public class Tutorial3 {
    public static void run() {
        System.out.println("Starting Tutorial3...");
        Network net = new Network();

        // load the network created by Tutorial1
        net.readFile("tutorial1.xdsl");
    }
}

```

```

        for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h)) {
            printNodeInfo(net, h);
        }

        System.out.println("Tutorial3 complete.");
    }

    private static void printNodeInfo(Network net, int nodeHandle) {
        System.out.printf("Node id/name: %s/%s\n",
            net.getNodeId(nodeHandle),
            net.getNodeName(nodeHandle));

        System.out.print("  Outcomes:");
        for (String outcomeId: net.getOutcomeIds(nodeHandle)) {
            System.out.print(" " + outcomeId);
        }
        System.out.println();

        String[] parentIds = net.getParentIds(nodeHandle);
        if (parentIds.length > 0) {
            System.out.print("  Parents:");
            for (String parentId: parentIds) {
                System.out.print(" " + parentId);
            }
            System.out.println();
        }

        String[] childIds = net.getChildIds(nodeHandle);
        if (childIds.length > 0) {
            System.out.print("  Children:");
            for (String childId: childIds) {
                System.out.print(" " + childId);
            }
            System.out.println();
        }

        printCptMatrix(net, nodeHandle);
    }

    private static void printCptMatrix(Network net, int nodeHandle) {
        double[] cpt = net.getNodeDefinition(nodeHandle);
        int[] parents = net.getParents(nodeHandle);
        int dimCount = 1 + parents.length;

        int[] dimSizes = new int[dimCount];
        for (int i = 0; i < dimCount - 1; i++) {
            dimSizes[i] = net.getOutcomeCount(parents[i]);
        }
        dimSizes[dimSizes.length - 1] = net.getOutcomeCount(nodeHandle);

        int[] coords = new int[dimCount];
        for (int elemIdx = 0; elemIdx < cpt.length; elemIdx++) {
            indexToCoords(elemIdx, dimSizes, coords);

            String outcome = net.getOutcomeId(nodeHandle, coords[dimCount - 1]);
            System.out.printf("    P(%s", outcome);

            if (dimCount > 1) {
                System.out.print(" | ");
                for (int parentIdx = 0; parentIdx < parents.length; parentIdx++)
                    {

```

```

        if (parentIdx > 0) System.out.print(",");
        int parentHandle = parents[parentIdx];
        System.out.printf("%s=%s",
            net.getNodeId(parentHandle),
            net.getOutcomeId(parentHandle, coords[parentIdx]));
    }
}

double prob = cpt[elemIdx];
System.out.printf(")=%f\n", prob);
}
}

private static void indexToCoords(int index, int[] dimSizes, int[] coords) {
    int prod = 1;
    for (int i = dimSizes.length - 1; i >= 0; i --) {
        coords[i] = (index / prod) % dimSizes[i];
        prod *= dimSizes[i];
    }
}
}

```

7.3.3 Tutorial3.cs

```

using System;
using Smile;

// Tutorial3 loads the XDSL file and prints the information
// about the structure (nodes and arcs) and the parameters
// (conditional probabilities of the nodes) of the network.

namespace SmileNetTutorial
{
    class Tutorial3
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial3...");
            Network net = new Network();

            // load the network created by Tutorial1
            net.ReadFile("tutorial1.xdsl");

            for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
            {
                PrintNodeInfo(net, h);
            }

            Console.WriteLine("Tutorial3 complete.");
        }

        private static void PrintNodeInfo(Network net, int nodeHandle)
        {
            Console.WriteLine("Node id/name: {0}/{1}",
                net.GetNodeId(nodeHandle),
                net.GetNodeName(nodeHandle));

            Console.Write(" Outcomes:");
            foreach (String outcomeId in net.GetOutcomeIds(nodeHandle))
            {

```

```

        Console.Write(" " + outcomeId);
    }
    Console.WriteLine();

    String[] parentIds = net.GetParentIds(nodeHandle);
    if (parentIds.Length > 0)
    {
        Console.Write("  Parents:");
        foreach (String parentId in parentIds)
        {
            Console.Write(" " + parentId);
        }
        Console.WriteLine();
    }

    String[] childIds = net.GetChildIds(nodeHandle);
    if (childIds.Length > 0)
    {
        Console.Write("  Children:");
        foreach (String childId in childIds)
        {
            Console.Write(" " + childId);
        }
        Console.WriteLine();
    }

    PrintCptMatrix(net, nodeHandle);
}

private static void PrintCptMatrix(Network net, int nodeHandle)
{
    double[] cpt = net.GetNodeDefinition(nodeHandle);
    int[] parents = net.GetParents(nodeHandle);
    int dimCount = 1 + parents.Length;

    int[] dimSizes = new int[dimCount];
    for (int i = 0; i < dimCount - 1; i++)
    {
        dimSizes[i] = net.GetOutcomeCount(parents[i]);
    }
    dimSizes[dimSizes.Length - 1] = net.GetOutcomeCount(nodeHandle);

    int[] coords = new int[dimCount];
    for (int elemIdx = 0; elemIdx < cpt.Length; elemIdx++)
    {
        IndexToCoords(elemIdx, dimSizes, coords);

        String outcome = net.GetOutcomeId(nodeHandle, coords[dimCount - 1]);
        Console.Write("    P({0}", outcome);

        if (dimCount > 1)
        {
            Console.Write(" | ");
            for (int pIdx = 0; pIdx < parents.Length; pIdx++)
            {
                if (pIdx > 0) Console.Write(",");
                int parentHandle = parents[pIdx];
                Console.Write("{0}={1}",
                    net.GetNodeId(parentHandle),
                    net.GetOutcomeId(parentHandle, coords[pIdx]));
            }
        }
    }
}

```

```

    }

    double prob = cpt[elemIdx];
    Console.WriteLine("{0}", prob);
  }
}

private static void IndexToCoords(
    int index, int[] dimSizes, int[] coords)
{
    int prod = 1;
    for (int i = dimSizes.Length - 1; i >= 0; i--)
    {
        coords[i] = (index / prod) % dimSizes[i];
        prod *= dimSizes[i];
    }
}
}
}

```

7.3.4 Tutorial3.R

```

library(rSMILE)
source("License.R")

# Tutorial3 loads the XDSL file and prints the information
# about the structure (nodes and arcs) and the parameters
# (conditional probabilities of the nodes) of the network.

indexToCoords = function(index, dimSizes) {
  prod <- 1L
  coords <- integer(length=length(dimSizes))
  for (i in length(dimSizes):1) {
    coords[i] <- floor(index / prod) %% dimSizes[[i]]
    prod <- prod * dimSizes[[i]]
  }
  return(coords)
}

printCptMatrix = function(net, nodeHandle) {
  cpt <- net$getNodeDefinition(nodeHandle)
  parents <- net$getParents(nodeHandle)
  dimCount <- 1 + length(parents)

  dimSizes <- sapply(parents, function(x) net$getOutcomeCount(x))
  dimSizes[length(dimSizes)+1] <- net$getOutcomeCount(nodeHandle)

  for (elemIdx in 0:(length(cpt)-1)) {
    coords <- indexToCoords(elemIdx, dimSizes)
    outcome <- net$getOutcomeId(nodeHandle, coords[dimCount])
    cat("    P(", outcome, sep="");
    if (dimCount > 1) {
      cat(" | ")
      parentIds <- sapply(parents, function(x) net$getNodeId(x))
      outcomeIds <- sapply(1:length(parents),
        function(x) net$getOutcomeId(parents[x], coords[x]))

      cat(paste(parentIds, outcomeIds, sep="=", collapse=","))
    }
    prob <- cpt[elemIdx + 1]
  }
}

```

```

        cat(sprintf(")=%f\n", prob))
    }
}

printNodeInfo = function(net, nodeHandle) {
  cat(sprintf("Node id/name: %s/%s\n",
    net$getNodeId(nodeHandle),
    net$getNodeName(nodeHandle)))
  cat("  Outcomes: ")
  cat(paste(net$getOutcomeIds(nodeHandle), collapse=" "))
  cat("\n")

  parentIds <- net$getParentIds(nodeHandle)
  if (length(parentIds) > 0) {
    cat("  Parents: ")
    cat(paste(parentIds, collapse=" "))
    cat("\n")
  }

  childIds <- net$getChildIds(nodeHandle)
  if (length(childIds) > 0) {
    cat("  Children: ")
    cat(paste(childIds, collapse=" "))
    cat("\n")
  }

  printCptMatrix(net, nodeHandle)
}

cat("Starting Tutorial3...\n")
net <- Network()

# load the network created by Tutorial1
net$readFile("tutorial1.xdsl")

nodes <- net$getAllNodes()

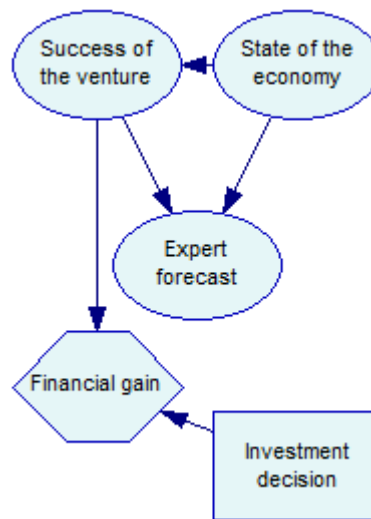
for (h in nodes) {
  printNodeInfo(net, h)
}

cat("Tutorial3 complete.\n")

```

7.4 Tutorial 4: Creating the Influence Diagram

We will further expand the model created in [Tutorial 1](#)⁸⁴ and turn it into an influence diagram. To this effect, we will add a decision node *Investment decision* and a utility node *Financial gain*. The decision will have two possible states: *Invest* and *DoNotInvest*, which will be the two decision options under consideration. Which option is chosen will impact the financial gain and this will be reflected by a directed arc from *Investment decision* to *Financial gain*. Whether the venture succeeds or fails will also impact the financial gain and this will be also reflected by a directed arc from *Success of the venture* to *Financial gain*.



We will show how to create this model. In the subsequent tutorial, we will show how to enter observations (evidence), how to perform inference, and how to retrieve the utilities calculated for the *Financial gain* node.

The program starts by reading the file, just like [Tutorial 2](#)^[95] and [Tutorial 3](#)^[101]. We convert the identifier of the node *Success* to node handle, which we will use later to create an arc between *Success* and *Gain*. Two new nodes will be created by calling a `createNode` helper function, which is slightly modified version of the `createCptNode` from [Tutorial 1](#)^[84]. The difference is that we now want to create different types of nodes. Therefore, `createNode` has one additional input parameter, an integer which specifies the node type. Another difference is that `createNode` needs to be able to add utility nodes, which do not have outcomes. The function checks for the value of its outcomes input parameter and if it is null, the outcome initialization is skipped.

`createNode` is called to add two new nodes:

- a decision node *Investment*, with a `Network.NodeType.DECISION` type identifier
- an utility node *Gain*, with a `Network.NodeType.UTILITY` type identifier

With new nodes created we connect them to the rest of the model with `Network.addArc`. The only thing left is the initialization of the parameters for the *Gain* node (*Invest* is a decision node and decision nodes have no numeric parameters). *Gain* has two parents with two outcomes each and size of its definition is $2 \times 2 \times 1 = 4$. The program therefore specifies four numbers for the utilities.

Python:

```

gain_definition = [
    10000, # Utility(Invest=I, Success=S)
    -5000, # Utility(Invest=I, Success=F)
    500,   # Utility(Invest=D, Success=S)
    500    # Utility(Invest=D, Success=F)
]
net.set_node_definition(g, gain_definition)

```

Java:

```

double[] gainDefinition = new double[] {
    10000, // Utility(Invest=I, Success=S)
    -5000, // Utility(Invest=I, Success=F)
    500,   // Utility(Invest=D, Success=S)
    500    // Utility(Invest=D, Success=F)
};
net.setNodeDefinition(g, gainDefinition);

```

C#:

```
double[] gainDefinition = new double[]
{
    10000, // Utility(Invest=I, Success=S)
    -5000, // Utility(Invest=I, Success=F)
    500,   // Utility(Invest=D, Success=S)
    500    // Utility(Invest=D, Success=F)
};
net.SetNodeDefinition(g, gainDefinition);
```

R:

```
gainDefinition <- c(10000, # Utility(Invest=I, Success=S)
                  -5000, # Utility(Invest=I, Success=F)
                  500,   # Utility(Invest=D, Success=S)
                  500)  # Utility(Invest=D, Success=F)
net$setNodeDefinition(g, gainDefinition)
```

The influence diagram is now complete, so we can write its contents to the file and exit the function. [Tutorial 5](#)¹¹⁵ will load the file and perform the inference.

7.4.1 Tutorial4.py

```
import pysmile

# Tutorial4 loads the XDSL file file created by Tutorial1
# and adds decision and utility nodes, which transforms
# a Bayesian Network (BN) into an Influence Diagram (ID).

class Tutorial4:
    def __init__(self):
        print("Starting Tutorial4...")
        net = pysmile.Network()

        net.read_file("tutorial1.xdsl")

        s = net.get_node("Success")
        i = self.create_node(net, pysmile.NodeType.DECISION,
                           "Invest", "Investment decision",
                           ["Invest", "DoNotInvest"], 160, 240)

        g = self.create_node(net, pysmile.NodeType.UTILITY,
                           "Gain", "Financial gain", None, 60, 200)

        net.add_arc(i, g)
        net.add_arc(s, g)

        gain_definition = [
            10000, # Utility(Invest=I, Success=S)
            -5000, # Utility(Invest=I, Success=F)
            500,   # Utility(Invest=D, Success=S)
            500    # Utility(Invest=D, Success=F)
        ]
        net.set_node_definition(g, gain_definition)

        net.write_file("tutorial4.xdsl")

        print("Tutorial4 complete: Influence diagram written to tutorial4.xdsl.")

    def create_node(self,
                    net, node_type, id, name,
                    outcomes, xPos, yPos):
```

```

handle = net.add_node(node_type, id)

net.set_node_name(handle, name)
net.set_node_position(handle, xPos, yPos, 85, 55)

if outcomes is not None:
    initial_outcome_count = net.get_outcome_count(handle)
    for i in range(0, initial_outcome_count):
        net.set_outcome_id(handle, i, outcomes[i])

    for i in range(initial_outcome_count, len(outcomes)):
        net.add_outcome(handle, outcomes[i])

return handle

```

7.4.2 Tutorial4.java

```

package tutorials;

import smile.*;

// Tutorial4 loads the XDSL file file created by Tutorial1
// and adds decision and utility nodes, which transforms
// a Bayesian Network (BN) into an Influence Diagram (ID).

public class Tutorial4 {
    public static void run() {
        System.out.println("Starting Tutorial4...");
        Network net = new Network();

        net.readFile("tutorial1.xdsl");

        int s = net.getNode("Success");

        int i = createNode(net, Network.NodeType.DECISION,
            "Invest", "Investment decision",
            new String[]{ "Invest", "DoNotInvest" }, 160, 240);

        int g = createNode(net, Network.NodeType.UTILITY,
            "Gain", "Financial gain", null, 60, 200);

        net.addArc(i, g);
        net.addArc(s, g);

        double[] gainDefinition = new double[] {
            10000, // Utility(Invest=I, Success=S)
            -5000, // Utility(Invest=I, Success=F)
            500,   // Utility(Invest=D, Success=S)
            500    // Utility(Invest=D, Success=F)
        };
        net.setNodeDefinition(g, gainDefinition);

        net.writeFile("tutorial4.xdsl");

        System.out.println("Tutorial4 complete: Influence diagram written to tutorial4.xdsl.");
    }

    private static int createNode(
        Network net, int nodeType, String id, String name,
        String[] outcomes, int xPos, int yPos) {

```

```

    int handle = net.addNode(nodeType, id);

    net.setNodeName(handle, name);
    net.setNodePosition(handle, xPos, yPos, 85, 55);

    if (outcomes != null) {
        int initialOutcomeCount = net.getOutcomeCount(handle);
        for (int i = 0; i < initialOutcomeCount; i++) {
            net.setOutcomeId(handle, i, outcomes[i]);
        }

        for (int i = initialOutcomeCount; i < outcomes.length; i++) {
            net.addOutcome(handle, outcomes[i]);
        }
    }

    return handle;
}
}

```

7.4.3 Tutorial4.cs

```

using System;
using Smile;

// Tutorial4 loads the XDSL file file created by Tutorial1
// and adds decision and utility nodes, which transforms
// a Bayesian Network (BN) into an Influence Diagram (ID).

namespace SmileNetTutorial
{
    class Tutorial4
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial4...");
            Network net = new Network();

            net.ReadFile("tutorial1.xdsl");

            int s = net.GetNode("Success");

            int i = CreateNode(net, Network.NodeType.Decision,
                "Invest", "Investment decision",
                new String[] { "Invest", "DoNotInvest" }, 160, 240);

            int g = CreateNode(net, Network.NodeType.Utility,
                "Gain", "Financial gain", null, 60, 200);

            net.AddArc(i, g);
            net.AddArc(s, g);

            double[] gainDefinition = new double[]
            {
                10000, // Utility(Invest=I, Success=S)
                -5000, // Utility(Invest=I, Success=F)
                500,   // Utility(Invest=D, Success=S)
                500    // Utility(Invest=D, Success=F)
            };
            net.SetNodeDefinition(g, gainDefinition);
        }
    }
}

```

```

        net.WriteFile("tutorial4.xdsl");

        Console.WriteLine("Tutorial4 complete: ID written to tutorial4.xdsl.");
    }

    private static int CreateNode(
        Network net, Network.NodeType nodeType, String id, String name,
        String[] outcomes, int xPos, int yPos)
    {
        int handle = net.AddNode(nodeType, id);

        net.SetNodeName(handle, name);
        net.SetNodePosition(handle, xPos, yPos, 85, 55);

        if (outcomes != null)
        {
            int initialOutcomeCount = net.GetOutcomeCount(handle);
            for (int i = 0; i < initialOutcomeCount; i++)
            {
                net.SetOutcomeId(handle, i, outcomes[i]);
            }

            for (int i = initialOutcomeCount; i < outcomes.Length; i++)
            {
                net.AddOutcome(handle, outcomes[i]);
            }
        }

        return handle;
    }
}

```

7.4.4 Tutorial4.R

```

library(rSMILE)
source("License.R")

# Tutorial4 loads the XDSL file file created by Tutorial1
# and adds decision and utility nodes, which transforms
# a Bayesian Network (BN) into an Influence Diagram (ID).

createNode = function(net, nodeType, id, name, outcomes, xPos, yPos) {
    handle <- net$addNode(nodeType, id)
    initialOutcomeCount <- net$getOutcomeCount(handle)
    count <- length(outcomes)
    net$setNodeName(handle, name)
    net$setNodePosition(handle, xPos, yPos, 85L, 55L)
    if (!is.null(outcomes)) {
        sapply(0:(initialOutcomeCount-1),
            function(x) net$setOutcomeId(handle, x, outcomes[x+1]))
        if (initialOutcomeCount < count) {
            sapply(initialOutcomeCount:(count-1),
                function(x) net$addOutcome(handle, outcomes[x+1]))
        }
    }

    return(handle)
}

```

```

cat("Starting Tutorial4...\n")
net <- Network()

net$readFile("tutorial1.xdsl")

s <- net$getNode("Success")

i <- createNode(net, net$NodeType$DECISION, "Invest", "Investment decision",
  c("Invest", "DoNotInvest"), 160L, 240L)

g <- createNode(net, net$NodeType$UTILITY, "Gain", "Financial Gain", NULL, 60, 200)

net$addArc(i, g)
net$addArc(s, g)

gainDefinition <- c(10000, # Utility(Invest=I, Success=S)
  -5000, # Utility(Invest=I, Success=F)
  500, # Utility(Invest=D, Success=S)
  500) # Utility(Invest=D, Success=F)

net$setNodeDefinition(g, gainDefinition)

net$writeFile("tutorial4.xdsl")

cat("Tutorial4 complete: Influence diagram written to tutorial4.xdsl.\n")

```

7.5 Tutorial 5: Inference in an Influence Diagram

This tutorial loads the influence diagram that we have created in [Tutorial 4](#)¹⁰⁹. We will perform multiple inference calls and display calculated utilities.

With model loaded, we calculate the probabilities and utilities by calling `Network.updateBeliefs`. A helper function, `printFinancialGain`, is called to print out the utilities stored in the *Gain* node. `printFinancialGain` uses `Network.getNodeValue` to obtain the utilities. To properly interpret the utilities, we also need to call `Network.getValueIndexingParents`, which is an array containing the handles of decision nodes not set to evidence.

Python:

```

expected_utility = net.get_node_value("Gain")
util_parents = net.get_value_indexing_parents("Gain")
self.print_gain_matrix(net, expected_utility, util_parents)

```

Java:

```

double[] expectedUtility = net.getNodeValue("Gain");
int[] utilParents = net.getValueIndexingParents("Gain");
printGainMatrix(net, expectedUtility, utilParents);

```

C#:

```

double[] expectedUtility = net.GetNodeValue("Gain");
int[] utilParents = net.GetValueIndexingParents("Gain");
printGainMatrix(net, expectedUtility, utilParents);

```

R:

```

expectedUtility <- net$getNodeValue("Gain")
utilParents <- net$getValueIndexingParents("Gain")
printGainMatrix(net, expectedUtility, utilParents)

```

`printGainMatrix` is a modification of `printCptMatrix` from [Tutorial 3](#)¹⁰¹. It iterates over the elements of an array and converts the linear index into multi-dimensional coordinates.

When we call `printGainMatrix` for the first time, the network has no evidence set. The utilities calculated without evidence suggest that we should not invest - here's what is printed on the screen:

```
Financial gain:
Utility(Invest=Invest)=-1850
Utility(Invest=DoNotInvest)=500
```

Next, we model the analyst's forecast to be good by calling local helper function `changeEvidenceAndUpdate`, which is based on the function with the same name from [Tutorial 2](#)⁹⁵.

Python:

```
self.change_evidence_and_update(net, "Forecast", "Good")
```

Java:

```
changeEvidenceAndUpdate(net, "Forecast", "Good");
```

C#:

```
ChangeEvidenceAndUpdate(net, "Forecast", "Good");
```

R:

```
changeEvidenceAndUpdate(net, "Forecast", "Good")
```

The expected gain changes, now the optimal decision is to invest:

```
Financial gain:
Utility(Invest=Invest)=4455.78
Utility(Invest=DoNotInvest)=500
```

Now we observe the state of the economy and conclude that it is growing - the program performs another `changeEvidenceAndUpdate` call. Growing economy makes our chances even better:

```
Financial gain:
Utility(Invest=Invest)=5000
Utility(Invest=DoNotInvest)=500
```

This concludes Tutorial 5.

7.5.1 Tutorial5.py

```
import pysmile

# Tutorial5 loads the XDSL file created by Tutorial4,
# then performs the series of inference calls,
# changing evidence each time.

class Tutorial5:
    def __init__(self):
        print("Starting Tutorial5...")
        net = pysmile.Network()

        net.read_file("tutorial4.xdsl")

        print("No evidence set.")
        net.update_beliefs()
        self.print_financial_gain(net)

        print("Setting Forecast=Good.")
        self.change_evidence_and_update(net, "Forecast", "Good")

        print("Adding Economy=Up")
        self.change_evidence_and_update(net, "Economy", "Up")

        print("Tutorial5 complete.")
```

```

def change_evidence_and_update(self, net, node_id, outcome_id):
    if outcome_id is not None:
        net.set_evidence(node_id, outcome_id)
    else:
        net.clear_evidence(node_id)

    net.update_beliefs()
    self.print_financial_gain(net)

def print_financial_gain(self, net):
    expected_utility = net.get_node_value("Gain")
    util_parents = net.get_value_indexing_parents("Gain")
    self.print_gain_matrix(net, expected_utility, util_parents)

def print_gain_matrix(self, net, mtx, parents):
    dim_count = 1 + len(parents)

    dim_sizes = [0] * dim_count
    for i in range(0, dim_count - 1):
        dim_sizes[i] = net.get_outcome_count(parents[i])
    dim_sizes[len(dim_sizes) - 1] = 1
    coords = [0] * dim_count
    for elem_idx in range(0, len(mtx)):
        self.index_to_coords(elem_idx, dim_sizes, coords)
        print("    Utility(", end="")
        if dim_count > 1:
            for parent_idx in range(0, len(parents)):
                if parent_idx > 0:
                    print(",", end="")
                parent_handle = parents[parent_idx]
                print(f"{net.get_node_id(parent_handle)}="
                    + f"{net.get_outcome_id(parent_handle, coords[parent_idx])}", end="")
            print(f")={mtx[elem_idx]}")
        print()

def index_to_coords(self, index, dim_sizes, coords):
    prod = 1
    for i in range(len(dim_sizes) - 1, -1, -1):
        coords[i] = int(index / prod) % dim_sizes[i]
        prod *= dim_sizes[i]

```

7.5.2 Tutorial5.java

```

package tutorials;

import smile.*;

// Tutorial5 loads the XDSL file created by Tutorial4,
// then performs the series of inference calls,
// changing evidence each time.

public class Tutorial5 {
    public static void run() {
        System.out.println("Starting Tutorial5...");
        Network net = new Network();

        net.readFile("tutorial4.xdsl");

        System.out.println("No evidence set.");
        net.updateBeliefs();
    }
}

```

```

    printFinancialGain(net);

    System.out.println("Setting Forecast=Good.");
    changeEvidenceAndUpdate(net, "Forecast", "Good");

    System.out.println("Adding Economy=Up");
    changeEvidenceAndUpdate(net, "Economy", "Up");

    System.out.println("Tutorial5 complete.");
}

static void changeEvidenceAndUpdate(
    Network net, String nodeId, String outcomeId) {
    if (outcomeId != null) {
        net.setEvidence(nodeId, outcomeId);
    } else {
        net.clearEvidence(nodeId);
    }

    net.updateBeliefs();
    printFinancialGain(net);
}

static void printFinancialGain(Network net) {
    double[] expectedUtility = net.getNodeValue("Gain");
    int[] utilParents = net.getValueIndexingParents("Gain");
    printGainMatrix(net, expectedUtility, utilParents);
}

static void printGainMatrix(Network net, double[] mtx, int[] parents) {
    int dimCount = 1 + parents.length;

    int[] dimSizes = new int[dimCount];
    for (int i = 0; i < dimCount - 1; i++) {
        dimSizes[i] = net.getOutcomeCount(parents[i]);
    }
    dimSizes[dimSizes.length - 1] = 1;

    int[] coords = new int[dimCount];
    for (int elemIdx = 0; elemIdx < mtx.length; elemIdx++) {
        indexToCoords(elemIdx, dimSizes, coords);

        System.out.print("    Utility(");

        if (dimCount > 1)
        {
            for (int parentIdx = 0; parentIdx < parents.length; parentIdx++)
            {
                if (parentIdx > 0) System.out.print(",");
                int parentHandle = parents[parentIdx];
                System.out.printf("%s=%s",
                    net.getNodeId(parentHandle),
                    net.getOutcomeId(parentHandle, coords[parentIdx]));
            }
        }

        System.out.printf(")=%f\n", mtx[elemIdx]);
    }
    System.out.println();
}

```

```

static void indexToCoords(int index, int[] dimSizes, int[] coords) {
    int prod = 1;
    for (int i = dimSizes.length - 1; i >= 0; i --) {
        coords[i] = (index / prod) % dimSizes[i];
        prod *= dimSizes[i];
    }
}
}

```

7.5.3 Tutorial5.cs

```

using System;
using Smile;

// Tutorial5 loads the XDSL file created by Tutorial4,
// then performs the series of inference calls,
// changing evidence each time.

namespace SmileNetTutorial
{
    class Tutorial5
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial5...");
            Network net = new Network();

            net.ReadFile("tutorial4.xdsl");

            Console.WriteLine("No evidence set.");
            net.UpdateBeliefs();
            PrintFinancialGain(net);

            Console.WriteLine("Setting Forecast=Good.");
            ChangeEvidenceAndUpdate(net, "Forecast", "Good");

            Console.WriteLine("Adding Economy=Up");
            ChangeEvidenceAndUpdate(net, "Economy", "Up");

            Console.WriteLine("Tutorial5 complete.");
        }

        static void ChangeEvidenceAndUpdate(Network net, String nodeId, String outcomeId)
        {
            if (outcomeId != null)
            {
                net.SetEvidence(nodeId, outcomeId);
            }
            else
            {
                net.ClearEvidence(nodeId);
            }
            net.UpdateBeliefs();
            PrintFinancialGain(net);
        }

        static void PrintFinancialGain(Network net)
        {
            double[] expectedUtility = net.GetNodeValue("Gain");
            int[] utilParents = net.GetValueIndexingParents("Gain");
        }
    }
}

```

```

        PrintGainMatrix(net, expectedUtility, utilParents);
    }

    static void PrintGainMatrix(Network net, double[] mtx, int[] parents)
    {
        int dimCount = 1 + parents.Length;

        int[] dimSizes = new int[dimCount];
        for (int i = 0; i < dimCount - 1; i++)
        {
            dimSizes[i] = net.GetOutcomeCount(parents[i]);
        }
        dimSizes[dimSizes.Length - 1] = 1;

        int[] coords = new int[dimCount];
        for (int elemIdx = 0; elemIdx < mtx.Length; elemIdx++)
        {
            IndexToCoords(elemIdx, dimSizes, coords);
            Console.Write("    Utility(");
            if (dimCount > 1)
            {
                for (int pIdx = 0; pIdx < parents.Length; pIdx++)
                {
                    if (pIdx > 0) Console.Write(",");
                    int parentHandle = parents[pIdx];
                    Console.Write("{0}={1}",
                        net.GetNodeId(parentHandle),
                        net.GetOutcomeId(parentHandle, coords[pIdx]));
                }
            }
            Console.Write(")={0}\n", mtx[elemIdx]);
        }
        Console.WriteLine();
    }

    static void IndexToCoords(int index, int[] dimSizes, int[] coords)
    {
        int prod = 1;
        for (int i = dimSizes.Length - 1; i >= 0; i--)
        {
            coords[i] = (index / prod) % dimSizes[i];
            prod *= dimSizes[i];
        }
    }
}

```

7.5.4 Tutorial5.R

```

library(rSMILE)
source("License.R")

# Tutorial5 loads the XDSL file created by Tutorial4,
# then performs the series of inference calls,
# changing evidence each time.

indexToCoords = function(index, dimSizes) {
    prod <- 1L
    coords <- integer(length=length(dimSizes))
    for (i in length(dimSizes):1) {

```

```

        coords[i] <- floor(index / prod) %% dimSizes[[i]]
        prod <- prod * dimSizes[[i]]
    }
    return(coords)
}

printGainMatrix = function(net, mtx, parents) {
    dimCount <- 1 + length(parents)

    dimSizes <- sapply(parents, function(x) net$getOutcomeCount(x))
    dimSizes[length(dimSizes)+1] <- 1

    for (elemIdx in 0:(length(mtx)-1)) {
        coords <- indexToCoords(elemIdx, dimSizes)
        cat("    Utility(");
        if (dimCount > 1) {
            parentIds <- sapply(parents, function(x) net$getNodeId(x))
            outcomeIds <- sapply(1:length(parents),
                                function(x) net$getOutcomeId(parents[x], coords[x]))

            cat(paste(parentIds, outcomeIds, sep="=", collapse=","))
        }
        cat(sprintf(")=%f\n", mtx[elemIdx+1]))
    }
}

printFinancialGain = function(net) {
    expectedUtility <- net$getNodeValue("Gain")
    utilParents <- net$getValueIndexingParents("Gain")
    printGainMatrix(net, expectedUtility, utilParents)
}

changeEvidenceAndUpdate = function(net, nodeId, outcomeId) {
    if (!is.null(outcomeId)) {
        net$setEvidence(nodeId, outcomeId)
    } else {
        net$clearEvidence(nodeId)
    }
    net$updateBeliefs()
    printFinancialGain(net)
}

cat("Starting Tutorial5...\n")
net <- Network()

net$readFile("tutorial4.xdsl")

cat("No evidence set.\n")
net$updateBeliefs()
printFinancialGain(net)

cat("Setting Forecast=Good.\n")
changeEvidenceAndUpdate(net, "Forecast", "Good")

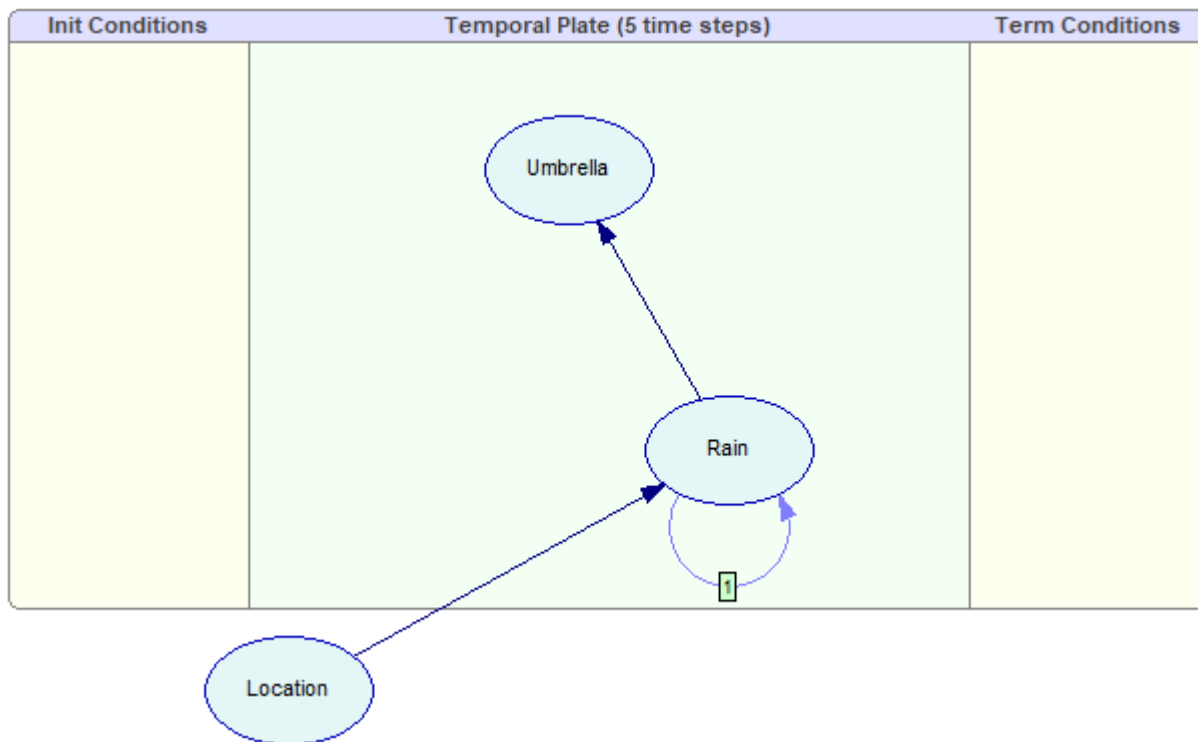
cat("Adding Economy=Up.\n")
changeEvidenceAndUpdate(net, "Economy", "Up")

cat("Tutorial5 complete.\n")

```

7.6 Tutorial 6: Dynamic model

Consider the following example, inspired by (Russell & Norvig, 1995), in which a security guard at some secret underground installation works on a shift of seven days and wants to know whether it is raining on the day of her return to the outside world. Her only access to the outside world occurs each morning when she sees the director coming in, with or without an umbrella. Furthermore, she knows that the government has two secret underground installations: one in Pittsburgh and one in the Sahara, but she does not know which one she is guarding. For each day t , the set of evidence contains a single variable $Umbrella_t$ (observation of an umbrella carried by the director) and the set of unobservable variables contains $Rain_t$ (a propositional variable with two states *true* and *false*, denoting whether it is raining) and $Location$ (with two possible states: *Pittsburgh* and *Sahara*). The prior probability of rain depends on the geographical location and on whether it rained on the previous day. For simplicity, we do not use the initial and terminal condition nodes in this tutorial.



The model contains three discrete chance nodes (CPT), created with a call to `createCptNode` helper function, just like in nodes in [Tutorial 1](#)⁸⁴. The *Rain* and *Umbrella* nodes are marked as belonging to the temporal plate by calling `Network.SetNodeTemporalType`:

Python:

```
net.set_node_temporal_type(rain, pysmile.NodeTemporalType.PLATE)
net.set_node_temporal_type(umb, pysmile.NodeTemporalType.PLATE)
```

Java:

```
net.setNodeTemporalType(rain, Network.NodeTemporalType.PLATE);
net.setNodeTemporalType(umb, Network.NodeTemporalType.PLATE);
```

C#:

```
net.SetNodeTemporalType(rain, Network.NodeTemporalType.Plate);
net.SetNodeTemporalType(umb, Network.NodeTemporalType.Plate);
```

R:

```
net$setNodeTemporalType(rain, net$NodeTemporalType$PLATE)
net$setNodeTemporalType(umb, net$NodeTemporalType$PLATE)
```

Two of the three arcs in the model are created by `Network.addArc`. The temporal arc expressing the dependency of $Rain_t$ on $Rain_{t-1}$ is created by the `Network.addTemporalArc` method. Please note the temporal order of the arc passed as the 3rd parameter:

Python:

```
net.add_arc(loc, rain)
net.add_temporal_arc(rain, rain, 1)
net.add_arc(rain, umb)
```

Java:

```
net.addArc(loc, rain);
net.addTemporalArc(rain, rain, 1);
net.addArc(rain, umb);
```

C#:

```
net.AddArc(loc, rain);
net.AddTemporalArc(rain, rain, 1);
net.AddArc(rain, umb);
```

R:

```
net$addArc(loc, rain)
net$addTemporalArc(rain, rain, 1)
net$addArc(rain, umb)
```

CPTs for the nodes are initialized with `Network.setNodeDefinition`, as in [Tutorial 1](#)⁸⁴. However, the node Rain requires **two** CPTs, because it has an incoming temporal arc of order 1. The additional CPT is set by the call to `Network.setNodeTemporalDefinition`:

Python:

```
rain_def_temporal = [
    0.7, # P(Rain=true | Location=Pittsburgh, Rain[t-1]=true)
    0.3, # P(Rain=false | Location=Pittsburgh, Rain[t-1]=true)
    0.3, # P(Rain=true | Location=Pittsburgh, Rain[t-1]=false)
    0.7, # P(Rain=false | Location=Pittsburgh, Rain[t-1]=false)
    0.001, # P(Rain=true | Location=Sahara, Rain[t-1]=true)
    0.999, # P(Rain=false | Location=Sahara, Rain[t-1]=true)
    0.01, # P(Rain=true | Location=Sahara, Rain[t-1]=false)
    0.99, # P(Rain=false | Location=Sahara, Rain[t-1]=false)
]
net.set_node_temporal_definition(rain, 1, rain_def_temporal)
```

Java:

```
double[] rainDefTemporal = new double[] {
    0.7, // P(Rain=true | Location=Pittsburgh, Rain[t-1]=true)
    0.3, // P(Rain=false | Location=Pittsburgh, Rain[t-1]=true)
    0.3, // P(Rain=true | Location=Pittsburgh, Rain[t-1]=false)
    0.7, // P(Rain=false | Location=Pittsburgh, Rain[t-1]=false)
    0.001, // P(Rain=true | Location=Sahara, Rain[t-1]=true)
    0.999, // P(Rain=false | Location=Sahara, Rain[t-1]=true)
    0.01, // P(Rain=true | Location=Sahara, Rain[t-1]=false)
    0.99, // P(Rain=false | Location=Sahara, Rain[t-1]=false)
};
net.setNodeTemporalDefinition(rain, 1, rainDefTemporal);
```

C#:

```
double[] rainDefTemporal = new double[]
{
    0.7, // P(Rain=true | Location=Pittsburgh, Rain[t-1]=true)
    0.3, // P(Rain=false | Location=Pittsburgh, Rain[t-1]=true)
    0.3, // P(Rain=true | Location=Pittsburgh, Rain[t-1]=false)
    0.7, // P(Rain=false | Location=Pittsburgh, Rain[t-1]=false)
    0.001, // P(Rain=true | Location=Sahara, Rain[t-1]=true)
    0.999, // P(Rain=false | Location=Sahara, Rain[t-1]=true)
    0.01, // P(Rain=true | Location=Sahara, Rain[t-1]=false)
    0.99, // P(Rain=false | Location=Sahara, Rain[t-1]=false)
}
```

```

    0.99 // P(Rain=false|Location=Sahara,Rain[t-1]=false)
};
net.SetNodeTemporalDefinition(rain, 1, rainDefTemporal);
R:
rainDefTemporal <- c( 0.7,
                      0.3,
                      0.3,
                      0.7,
                      0.001,
                      0.999,
                      0.01,
                      0.99 )

net$setNodeTemporalDefinition(rain, 1, rainDefTemporal)

```

Finally, we adjust the number of slices created during the network unrolling with a call to `Network.setSliceCount`. The network is complete. The program proceeds to the inference, first without any evidence in the network, then with two observations of the *Umbrella*, set in the time steps $t=1$ and $t=3$:

Python:

```

net.set_temporal_evidence(umb, 1, 0)
net.set_temporal_evidence(umb, 3, 1)

```

Java:

```

net.setTemporalEvidence(umb, 1, 0);
net.setTemporalEvidence(umb, 3, 1);

```

C#:

```

net.SetTemporalEvidence(umb, 1, 0);
net.SetTemporalEvidence(umb, 3, 1);

```

R:

```

net$setTemporalEvidence(umb, 1, 0)
net$setTemporalEvidence(umb, 3, 1)

```

In dynamic Bayesian networks, the plate nodes have their beliefs calculated for each time slice. The number of elements in the node value matrix for these nodes is a product of outcome count and slice count. The helper function `updateAndShowTemporalResults` iterates over this matrix using two nested loops in order to print the results. Since the probabilities for the same slice are adjacent, the inner loop uses the product of the outcome count and slice index from the outer loop as a base index.

Python:

```

v = net.get_node_value(h)
for slice_idx in range(0, slice_count):
    s = "\tt=" + str(slice_idx) + ":"
    for i in range(0, outcome_count):
        s = s + " " + str(v[slice_idx * outcome_count + i])
    print(s)

```

Java:

```

double[] v = net.getNodeValue(h);
for (int sliceIdx = 0; sliceIdx < sliceCount; sliceIdx++) {
    System.out.printf("\tt=%d:", sliceIdx);
    for (int i = 0; i < outcomeCount; i++) {
        System.out.printf(" %f", v[sliceIdx * outcomeCount + i]);
    }
    System.out.println();
}

```

C#:

```

double[] v = net.GetNodeValue(h);
for (int sliceIdx = 0; sliceIdx < sliceCount; sliceIdx++)
{

```

```

        Console.WriteLine("\tt={0}:", sliceIdx);
        for (int i = 0; i < outcomeCount; i++)
        {
            Console.WriteLine(" {0}", v[sliceIdx * outcomeCount + i]);
        }
        Console.WriteLine();
    }
}
R:
v <- net$getNodeValue(h)
for (sliceIdx in 0:(sliceCount-1)) {
    cat(sprintf("\tt=%d:", sliceIdx))
    for (i in 0:(outcomeCount-1)) {
        cat(sprintf(" %f", v[(sliceIdx*outcomeCount+i)+1]))
    }
    cat("\n")
}

```

7.6.1 Tutorial6.py

```

import pysmile

# Tutorial6 creates a dynamic Bayesian network (DBN),
# performs the inference, then saves the model to disk.

class Tutorial6:
    def __init__(self):
        print("Starting Tutorial6...")
        net = pysmile.Network()
        loc = self.create_cpt_node(net,
            "Location", "Location",
            ["Pittsburgh", "Sahara"],
            160, 360)
        rain = self.create_cpt_node(net,
            "Rain", "Rain",
            ["true", "false"],
            380, 240)
        umb = self.create_cpt_node(net,
            "Umbrella", "Umbrella",
            ["true", "false"],
            300, 100)

        net.set_node_temporal_type(rain, pysmile.NodeTemporalType.PLATE)
        net.set_node_temporal_type(umb, pysmile.NodeTemporalType.PLATE)

        net.add_arc(loc, rain)
        net.add_temporal_arc(rain, rain, 1)
        net.add_arc(rain, umb)

        rain_def = [
            0.7, # P(Rain=true | Location=Pittsburgh)
            0.3, # P(Rain=false | Location=Pittsburgh)
            0.01, # P(Rain=true | Location=Sahara)
            0.99 # P(Rain=false | Location=Sahara)
        ]

        net.set_node_definition(rain, rain_def)

        rain_def_temporal = [

```

```

    0.7, # P(Rain=true | Location=Pittsburgh, Rain[t-1]=true)
    0.3, # P(Rain=false | Location=Pittsburgh, Rain[t-1]=true)
    0.3, # P(Rain=true | Location=Pittsburgh, Rain[t-1]=false)
    0.7, # P(Rain=false | Location=Pittsburgh, Rain[t-1]=false)
    0.001, # P(Rain=true | Location=Sahara, Rain[t-1]=true)
    0.999, # P(Rain=false | Location=Sahara, Rain[t-1]=true)
    0.01, # P(Rain=true | Location=Sahara, Rain[t-1]=false)
    0.99 # P(Rain=false | Location=Sahara, Rain[t-1]=false)
]
net.set_node_temporal_definition(rain, 1, rain_def_temporal)

umb_def = [
    0.9, # P(Umbrella=true | Rain=true)
    0.1, # P(Umbrella=false | Rain=true)
    0.2, # P(Umbrella=true | Rain=false)
    0.8 # P(Umbrella=false | Rain=false)
]
net.set_node_definition(umb, umb_def)

net.set_slice_count(5)

print("Performing update without evidence.")
self.update_and_show_temporal_results(net)

print("Setting Umbrella[t=1], to true and Umbrella[t=3] to false.")
net.set_temporal_evidence(umb, 1, 0)
net.set_temporal_evidence(umb, 3, 1)
self.update_and_show_temporal_results(net)

net.write_file("tutorial6.xdsl")
print("Tutorial6 complete: Network written to tutorial6.xdsl")

def update_and_show_temporal_results(self, net):
    net.update_beliefs()
    slice_count = net.get_slice_count()
    for h in net.get_all_nodes():
        if net.get_node_temporal_type(h) == pysmile.NodeTemporalType.PLATE:
            outcome_count = net.get_outcome_count(h)
            v = net.get_node_value(h)
            print(f"Temporal beliefs for {net.get_node_id(h)}:")
            for slice_idx in range(0, slice_count):
                print(f"\ttt={slice_idx}:", end="")
                for i in range(0, outcome_count):
                    print(f" {v[slice_idx * outcome_count + i]}", end="")
                print()

def create_cpt_node(self, net, id, name, outcomes, x_pos, y_pos):
    handle = net.add_node(pysmile.NodeType.CPT, id)

    net.set_node_name(handle, name)
    net.set_node_position(handle, x_pos, y_pos, 85, 55)

    initial_outcome_count = net.get_outcome_count(handle)

    for i in range(0, initial_outcome_count):
        net.set_outcome_id(handle, i, outcomes[i])

    for i in range(initial_outcome_count, len(outcomes)):
        net.add_outcome(handle, outcomes[i])

    return handle

```

7.6.2 Tutorial6.java

```

package tutorials;

import smile.*;

// Tutorial6 creates a dynamic Bayesian network (DBN),
// performs the inference, then saves the model to disk.

public class Tutorial6 {
    public static void run() {
        System.out.println("Starting Tutorial6...");
        Network net = new Network();

        int loc = createCptNode(
            net, "Location", "Location",
            new String[] { "Pittsburgh", "Sahara" },
            160, 360);

        int rain = createCptNode(
            net, "Rain", "Rain",
            new String[] { "true", "false" },
            380, 240);

        int umb = createCptNode(
            net, "Umbrella", "Umbrella",
            new String[] { "true", "false" },
            300, 100);

        net.setNodeTemporalType(rain, Network.NodeTemporalType.PLATE);
        net.setNodeTemporalType(umb, Network.NodeTemporalType.PLATE);

        net.addArc(loc, rain);
        net.addTemporalArc(rain, rain, 1);
        net.addArc(rain, umb);

        double[] rainDef = new double[] {
            0.7, // P(Rain=true | Location=Pittsburgh)
            0.3, // P(Rain=false | Location=Pittsburgh)
            0.01, // P(Rain=true | Location=Sahara)
            0.99 // P(Rain=false | Location=Sahara)
        };
        net.setNodeDefinition(rain, rainDef);

        double[] rainDefTemporal = new double[] {
            0.7, // P(Rain=true | Location=Pittsburgh, Rain[t-1]=true)
            0.3, // P(Rain=false | Location=Pittsburgh, Rain[t-1]=true)
            0.3, // P(Rain=true | Location=Pittsburgh, Rain[t-1]=false)
            0.7, // P(Rain=false | Location=Pittsburgh, Rain[t-1]=false)
            0.001, // P(Rain=true | Location=Sahara, Rain[t-1]=true)
            0.999, // P(Rain=false | Location=Sahara, Rain[t-1]=true)
            0.01, // P(Rain=true | Location=Sahara, Rain[t-1]=false)
            0.99 // P(Rain=false | Location=Sahara, Rain[t-1]=false)
        };
        net.setNodeTemporalDefinition(rain, 1, rainDefTemporal);

        double[] umbDef = new double[] {
            0.9, // P(Umbrella=true | Rain=true)
            0.1, // P(Umbrella=false | Rain=true)
        };
    }
}

```

```

        0.2, // P(Umbrella=true | Rain=false)
        0.8 // P(Umbrella=false | Rain=false)
    };
    net.setNodeDefinition(umb, umbDef);

    net.setSliceCount(5);

    System.out.println("Performing update without evidence.");
    updateAndShowTemporalResults(net);

    System.out.println("Setting Umbrella[t=1] to true and Umbrella[t=3] to false.");
    net.setTemporalEvidence(umb, 1, 0);
    net.setTemporalEvidence(umb, 3, 1);
    updateAndShowTemporalResults(net);

    net.writeFile("tutorial6.xdsl");
    System.out.println("Tutorial6 complete: Network written to tutorial6.xdsl");
}

private static void updateAndShowTemporalResults(Network net) {
    net.updateBeliefs();
    int sliceCount = net.getSliceCount();
    for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h)) {
        if (net.getNodeTemporalType(h) == Network.NodeTemporalType.PLATE) {
            int outcomeCount = net.getOutcomeCount(h);
            System.out.printf(
                "Temporal beliefs for %s:\n", net.getNodeId(h));
            double[] v = net.getNodeValue(h);
            for (int sliceIdx = 0; sliceIdx < sliceCount; sliceIdx++) {
                System.out.printf("\ttt=%d:", sliceIdx);
                for (int i = 0; i < outcomeCount; i++) {
                    System.out.printf(" %f", v[sliceIdx*outcomeCount+i]);
                }
                System.out.println();
            }
        }
    }
    System.out.println();
}

private static int createCptNode(
    Network net, String id, String name,
    String[] outcomes, int xPos, int yPos) {
    int handle = net.addNode(Network.NodeType.CPT, id);

    net.setNodeName(handle, name);
    net.setNodePosition(handle, xPos, yPos, 85, 55);

    int initialOutcomeCount = net.getOutcomeCount(handle);
    for (int i = 0; i < initialOutcomeCount; i++) {
        net.setOutcomeId(handle, i, outcomes[i]);
    }

    for (int i = initialOutcomeCount; i < outcomes.length; i++) {
        net.addOutcome(handle, outcomes[i]);
    }

    return handle;
}
}

```

7.6.3 Tutorial6.cs

```
using System;
using Smile;

// Tutorial6 creates a dynamic Bayesian network (DBN),
// performs the inference, then saves the model to disk.

namespace SmileNetTutorial
{
    class Tutorial6
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial6...");
            Network net = new Network();

            int loc = CreateCptNode(
                net, "Location", "Location",
                new String[] { "Pittsburgh", "Sahara" },
                160, 360);

            int rain = CreateCptNode(
                net, "Rain", "Rain",
                new String[] { "true", "false" },
                380, 240);

            int umb = CreateCptNode(
                net, "Umbrella", "Umbrella",
                new String[] { "true", "false" },
                300, 100);

            net.SetNodeTemporalType(rain, Network.NodeTemporalType.Plate);
            net.SetNodeTemporalType(umb, Network.NodeTemporalType.Plate);

            net.AddArc(loc, rain);
            net.AddTemporalArc(rain, rain, 1);
            net.AddArc(rain, umb);

            double[] rainDef = new double[]
            {
                0.7, // P(Rain=true | Location=Pittsburgh)
                0.3, // P(Rain=false | Location=Pittsburgh)
                0.01, // P(Rain=true | Location=Sahara)
                0.99 // P(Rain=false | Location=Sahara)
            };
            net.SetNodeDefinition(rain, rainDef);

            double[] rainDefTemporal = new double[]
            {
                0.7, // P(Rain=true | Location=Pittsburgh, Rain[t-1]=true)
                0.3, // P(Rain=false | Location=Pittsburgh, Rain[t-1]=true)
                0.3, // P(Rain=true | Location=Pittsburgh, Rain[t-1]=false)
                0.7, // P(Rain=false | Location=Pittsburgh, Rain[t-1]=false)
                0.001, // P(Rain=true | Location=Sahara, Rain[t-1]=true)
                0.999, // P(Rain=false | Location=Sahara, Rain[t-1]=true)
                0.01, // P(Rain=true | Location=Sahara, Rain[t-1]=false)
                0.99 // P(Rain=false | Location=Sahara, Rain[t-1]=false)
            };
            net.SetNodeTemporalDefinition(rain, 1, rainDefTemporal);
        }
    }
}
```

```

double[] umbDef = new double[]
{
    0.9, // P(Umbrella=true | Rain=true)
    0.1, // P(Umbrella=false | Rain=true)
    0.2, // P(Umbrella=true | Rain=false)
    0.8 // P(Umbrella=false | Rain=false)
};
net.SetNodeDefinition(umb, umbDef);

net.SetSliceCount(5);

Console.WriteLine("Performing update without evidence.");
UpdateAndShowTemporalResults(net);

Console.WriteLine("Setting Umbrella[t=1] to true and Umbrella[t=3] to false.");
net.SetTemporalEvidence(umb, 1, 0);
net.SetTemporalEvidence(umb, 3, 1);
UpdateAndShowTemporalResults(net);

net.WriteFile("tutorial6.xdsl");
Console.WriteLine("Tutorial6 complete: Network written to tutorial6.xdsl");
}

private static void UpdateAndShowTemporalResults(Network net)
{
    net.UpdateBeliefs();
    int sliceCount = net.GetSliceCount();
    for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
    {
        if (net.GetNodeTemporalType(h) == Network.NodeTemporalType.Plate)
        {
            int outcomeCount = net.GetOutcomeCount(h);
            Console.WriteLine("Temporal beliefs for {0}:", net.GetNodeId(h));
            double[] v = net.GetNodeValue(h);
            for (int sliceIdx = 0; sliceIdx < sliceCount; sliceIdx++)
            {
                Console.WriteLine("\ttt={0}:", sliceIdx);
                for (int i = 0; i < outcomeCount; i++)
                {
                    Console.WriteLine("\t\t{0} {1}", sliceIdx, v[sliceIdx * outcomeCount + i]);
                }
                Console.WriteLine();
            }
        }
    }
    Console.WriteLine();
}

private static int CreateCptNode(
    Network net, String id, String name,
    String[] outcomes, int xPos, int yPos)
{
    int handle = net.AddNode(Network.NodeType.Cpt, id);

    net.SetNodeName(handle, name);
    net.SetNodePosition(handle, xPos, yPos, 85, 55);

    int initialOutcomeCount = net.GetOutcomeCount(handle);
    for (int i = 0; i < initialOutcomeCount; i++)
    {

```

```

        net.SetOutcomeId(handle, i, outcomes[i]);
    }

    for (int i = initialOutcomeCount; i < outcomes.Length; i++)
    {
        net.AddOutcome(handle, outcomes[i]);
    }

    return handle;
}
}
}

```

7.6.4 Tutorial6.R

```

library(rSMILE)
source("License.R")

# Tutorial6 creates a dynamic Bayesian network (DBN),
# performs the inference, then saves the model to disk.

createCptNode = function(net, id, name, outcomes, xPos, yPos) {
    handle <- net$addNode(net$NodeType$CPT, id)
    initialOutcomeCount <- net$getOutcomeCount(handle)
    count <- length(outcomes)
    net$setNodeName(handle, name)
    net$setNodePosition(handle, xPos, yPos, 85L, 55L)
    if (!is.null(outcomes)) {
        sapply(0:(initialOutcomeCount-1),
            function(x) net$setOutcomeId(handle, x, outcomes[x+1]))
        if (initialOutcomeCount < count) {
            sapply(initialOutcomeCount:(count-1),
                function(x) net$addOutcome(handle, outcomes[x+1]))
        }
    }

    return(handle)
}

updateAndShowTemporalResults = function(net) {
    net$updateBeliefs()
    sliceCount <- net$getNodeSliceCount()
    handles <- net$getAllNodes()
    for (h in handles) {
        if (net$getNodeTemporalType(h) == net$NodeTemporalType$PLATE) {
            outcomeCount <- net$getNodeOutcomeCount(h)
            cat(sprintf("Temporal beliefs for %s:\n", net$getNodeId(h)))
            v <- net$getNodeValue(h)
            for (sliceIdx in 0:(sliceCount-1)) {
                cat(sprintf("\ttt=%d:", sliceIdx))
                for (i in 0:(outcomeCount-1)) {
                    cat(sprintf(" %f", v[(sliceIdx*outcomeCount+i)+1]))
                }
                cat("\n")
            }
        }
    }
    cat("\n")
}

```

```

cat("Starting Tutorial6...\n")
net <- Network()

loc <- createCptNode(net, "Location", "Location", c("Pittsburgh", "Sahara"), 160, 360)
rain <- createCptNode(net, "Rain", "Rain", c("true", "false"), 380, 240)
umb <- createCptNode(net, "Umbrella", "Umbrella", c("true", "false"), 300, 100)

net$setNodeTemporalType(rain, net$NodeTemporalType$PLATE)
net$setNodeTemporalType(umb, net$NodeTemporalType$PLATE)

net$addArc(loc, rain)
net$addTemporalArc(rain, rain, 1)
net$addArc(rain, umb)

rainDef <- c( 0.7,
              0.3,
              0.01,
              0.99 )

net$setNodeDefinition(rain, rainDef)

rainDefTemporal <- c( 0.7,
                     0.3,
                     0.3,
                     0.7,
                     0.001,
                     0.999,
                     0.01,
                     0.99 )

net$setNodeTemporalDefinition(rain, 1, rainDefTemporal)

umbDef <- c( 0.9,
            0.1,
            0.2,
            0.8 )

net$setNodeDefinition(umb, umbDef)

net$setSliceCount(5)

cat("Performing update without evidence.\n")
updateAndShowTemporalResults(net)

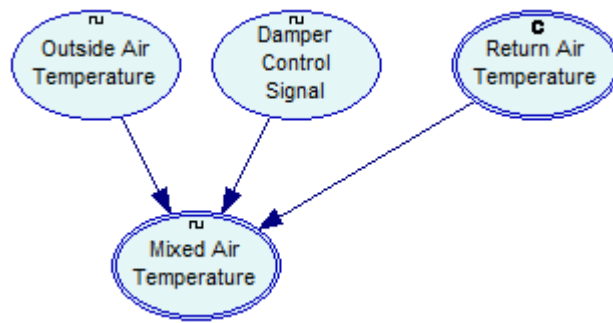
cat("Setting Umbrella[t=1] to true and Umbrella[t=3] to false.\n")
net$setTemporalEvidence(umb, 1, 0)
net$setTemporalEvidence(umb, 3, 1)
updateAndShowTemporalResults(net)

net$writeFile("tutorial6.xdsl")
cat("Tutorial6 complete: Network written to tutorial6.xdsl\n")

```

7.7 Tutorial 7: Continuous model

The continuous Bayesian network used in this tutorial focuses on a fragment of a forced air heating and cooling system. In order to improve the system's efficiency, return air is mixed with the air that is drawn from outside. Temperature of the outside air depends on the weather and is specified by means of a Normal distribution. Return air temperature is constant and depends on the thermostat setting. Damper control signal determines the composition of the mixture.



Temperature of the mixture is calculated according to the following equation: $tma = toa * u_d + (tra - tra * u_d)$, where tma is mixed air temperature, toa is outside air temperature, u_d is the damper signal, and tra is return air temperature.

The nodes in this model are created by the helper function `createEquationNode`. It is a modified version of the `createCptNode` function used in the previous tutorials. The bold text marks the difference between two functions.

Python:

```
def create_equation_node(self, net, id, name,
                        equation, lo_bound, hi_bound,
                        x_pos, y_pos):
    handle = net.add_node(pysmile.NodeType.EQUATION, id)
    net.set_node_name(handle, name)
    net.set_node_equation(handle, equation)
    net.set_node_equation_bounds(handle, lo_bound, hi_bound)
    net.set_node_position(handle, x_pos, y_pos, 85, 55)
    return handle
```

Java:

```
static int createEquationNode(
    Network net, String id, String name,
    String equation, double loBound, double hiBound,
    int xPos, int yPos) {
    int handle = net.addNode(Network.NodeType.EQUATION, id);
    net.setNodeName(handle, name);
    net.setNodeEquation(handle, equation);
    net.setNodeEquationBounds(handle, loBound, hiBound);
    net.setNodePosition(handle, xPos, yPos, 85, 55);
    return handle;
}
```

C#:

```
private static int CreateEquationNode(
    Network net, String id, String name,
    String equation, double loBound, double hiBound,
    int xPos, int yPos)
{
    int handle = net.AddNode(Network.NodeType.Equation, id);
    net.SetNodeName(handle, name);
    net.SetNodePosition(handle, xPos, yPos, 85, 55);
    net.SetNodeEquation(handle, equation);
    net.SetNodeEquationBounds(handle, loBound, hiBound);
    return handle;
}
```

R:

```
createEquationNode = function(net, id, name,
                             equation, loBound, hiBound,
```

```

        xPos, yPos) {
  handle <- net$addNode(net$NodeType$EQUATION, id)
  net$setNodeName(handle, name)
  net$setNodeEquation(handle, equation)
  net$setNodeEquationBounds(handle, loBound, hiBound)
  net$setNodePosition(handle, xPos, yPos, 85, 55)

  return(handle)
}

```

Instead of discrete outcomes, we specify the node equation and the bounds for the node value. Another helper function, `setUniformIntervals`, is used to define the discretization intervals. They will be used by the inference algorithm when the evidence is set for *Mixed Air Temperature* node (which has parents). The uniform intervals are chosen for simplicity here; in general case the choice of interval edges should be done based on the actual expected distribution of the node value (for example, in case of the Normal distribution, we might create narrow discretization intervals close to the mean.)

Note that while the model has three arcs, there are no calls to `Network.addArc` in this tutorial. The arcs are created implicitly by `Network.setNodeEquation` method (called by `createEquationNode` function).

The network is complete now and we can proceed to inference. Three inference calls are made, one without evidence and two with continuous evidence specified by calling `Network.setContEvidence`. Setting the *Outside Air Temperature* to 28.5 degrees (toa is the name of int variable holding the handle of the *Outside Air Temperature* node):

Python:

```
net.set_cont_evidence(toa, 28.5)
```

Java:

```
net.setContEvidence(toa, 28.5);
```

C#:

```
net.SetContEvidence(toa, 28.5);
```

R:

```
net$setContEvidence(toa, 28.5)
```

The program uses `updateAndShowStats` helper function for inference. The helper calls `Network.updateBeliefs` and iterates over the nodes in the network and calls another helper, `showStats`, for each node. `showStats` first checks if the node has evidence set. If it does, the evidence value is printed, and the function returns. If node has no evidence, we need to check if its value comes from the sampling or discretized inference.

If discretization was used, `Network.isValueDiscretized` returns true. In such case, the array returned from `Network.getNodeValue` contains the probability distribution over node discretization intervals. To display human-readable information about the intervals and probabilities, `showStats` calls `Network.getNodeEquationDiscretization` and `getNodeEquationBounds`. Note that the program setup these node attributes in `createEquationNode`.

If node value was sampled, `Network.isValueDiscretized` returns false. `Network.getNodeValue` returns the array of sampled node values. After sampling, it's also possible to obtain sample statistics (mean, standard deviation, min and max values) with a call to `Network.getNodeSampleStats`.

At the end of the tutorial, the model is saved to disk. [Tutorial 8](#)¹⁴² will expand it into a hybrid network by adding CPT nodes.

7.7.1 Tutorial7.py

```
import pysmile

# Tutorial7 creates a network with three equation-based nodes
# performs the inference, then saves the model to disk.

class Tutorial7:
    def __init__(self):
        print("Starting Tutorial7...")
        net = pysmile.Network()

        net.set_outlier_rejection_enabled(True)

        self.create_equation_node(net,
            "tra", "Return Air Temperature",
            "tra=24", 23.9, 24.1, 280, 100)

        self.create_equation_node(net,
            "u_d", "Damper Control Signal",
            "u_d=Bernoulli(0.539)*0.8+0.2", 0, 1, 160, 100)

        toa = self.create_equation_node(net,
            "toa", "Outside Air Temperature",
            "toa=Normal(11,15)", -10, 40, 60, 100)

        # tra, toa and u_d are referenced in equation
        # arcs are created automatically
        tma = self.create_equation_node(net,
            "tma", "Mixed Air Temperature",
            "tma=toa*u_d+(tra-tra*u_d)", 10, 30, 110, 200)

        self.set_uniform_intervals(net, toa, 5)
        self.set_uniform_intervals(net, tma, 4)

        print("Results with no evidence:")
        self.update_and_show_stats(net)

        net.set_cont_evidence(toa, 28.5)
        print("Results with outside air temperature set to 28.5:")
        self.update_and_show_stats(net)

        net.clear_evidence(toa)
        print("Results with mixed air temperature set to 21:")
        net.set_cont_evidence(tma, 21.0)
        self.update_and_show_stats(net)

        net.write_file("tutorial7.xdsl")
        print("Tutorial7 complete: Network written to tutorial7.xdsl")

    def create_equation_node(self, net, id, name, equation, lo_bound,
                            hi_bound, x_pos, y_pos):
        handle = net.add_node(pysmile.NodeType.EQUATION, id)
        net.set_node_name(handle, name)
        net.set_node_equation(handle, equation)
        net.set_node_equation_bounds(handle, lo_bound, hi_bound)

        net.set_node_position(handle, x_pos, y_pos, 85, 55)

        return handle
```

```

def show_stats(self, net, node_handle):
    node_id = net.get_node_id(node_handle)

    if net.is_evidence(node_handle):
        v = net.get_cont_evidence(node_handle)
        print(f"{node_id} has evidence set: {v}")
        return

    if net.is_value_discretized(node_handle):
        print(f"{node_id} is discretized.")
        iv = net.get_node_equation_discretization(node_handle)
        bounds = net.get_node_equation_bounds(node_handle)
        disc_beliefs = net.get_node_value(node_handle)
        lo = bounds[0]
        for i in range(0, len(disc_beliefs)):
            hi = iv[i].boundary
            print(f"\tP({node_id} in {lo}..{hi})={disc_beliefs[i]}")
            lo = hi
    else:
        stats = net.get_node_sample_stats(node_handle)
        print(f"{node_id}: mean={stats[0]} stddev={stats[1]} min={stats[2]} max={stats[3]}")

def set_uniform_intervals(self, net, node_handle, count):
    bounds = net.get_node_equation_bounds(node_handle)
    lo = bounds[0]
    hi = bounds[1]

    iv = [None] * count
    for i in range(0, count):
        iv[i] = pysmile.DiscretizationInterval("", lo + (i + 1) * (hi - lo) / count)

    net.set_node_equation_discretization(node_handle, iv)

def update_and_show_stats(self, net):
    net.update_beliefs()
    for h in net.get_all_nodes():
        self.show_stats(net, h)
    print()

```

7.7.2 Tutorial7.java

```

package tutorials;

import smile.*;

// Tutorial7 creates a network with three equation-based nodes
// performs the inference, then saves the model to disk.

public class Tutorial7 {
    public static void run() {
        System.out.println("Starting Tutorial7...");
        Network net = new Network();

        net.setOutlierRejectionEnabled(true);

        createEquationNode(net,
            "tra", "Return Air Temperature",
            "tra=24", 23.9, 24.1,
            280, 100);
    }
}

```

```

        createEquationNode(net,
            "u_d", "Damper Control Signal",
            "u_d = Bernoulli(0.539)*0.8 + 0.2", 0, 1,
            160, 100);

        int toa = createEquationNode(net,
            "toa", "Outside Air Temperature",
            "toa=Normal(11,15)", -10, 40,
            60, 100);

        // tra, toa and u_d are referenced in equation
        // arcs are created automatically
        int tma = createEquationNode(net,
            "tma", "Mixed Air Temperature",
            "tma=toa*u_d+(tra-tra*u_d)", 10, 30,
            110, 200);

        setUniformIntervals(net, toa, 5);
        setUniformIntervals(net, tma, 4);

        System.out.println("Results with no evidence:");
        updateAndShowStats(net);

        net.setContEvidence(toa, 28.5);
        System.out.println("Results with outside air temperature set to 28.5:");
        updateAndShowStats(net);

        net.clearEvidence(toa);
        System.out.println("Results with mixed air temperature set to 21:");
        net.setContEvidence(tma, 21.0);
        updateAndShowStats(net);

        net.writeFile("tutorial7.xdsl");
        System.out.println("Tutorial7 complete: Network written to tutorial7.xdsl");
    }

    static int createEquationNode(
        Network net, String id, String name,
        String equation, double loBound, double hiBound,
        int xPos, int yPos) {
        int handle = net.addNode(Network.NodeType.EQUATION, id);
        net.setNodeName(handle, name);
        net.setNodeEquation(handle, equation);
        net.setNodeEquationBounds(handle, loBound, hiBound);

        net.setNodePosition(handle, xPos, yPos, 85, 55);

        return handle;
    }

    static void showStats(Network net, int nodeHandle) {
        String nodeId = net.getNodeId(nodeHandle);

        if (net.isEvidence(nodeHandle)) {
            double v = net.getContEvidence(nodeHandle);
            System.out.printf("%s has evidence set (%g)\n", nodeId, v);
            return;
        }

        if (net.isValueDiscretized(nodeHandle)) {

```

```

        System.out.printf("%s is discretized.\n", nodeId);
        DiscretizationInterval[] iv =
net.getNodeEquationDiscretization(nodeHandle);
        double[] bounds = net.getNodeEquationBounds(nodeHandle);
        double[] discBeliefs = net.getNodeValue(nodeHandle);
        double lo = bounds[0];
        for (int i = 0; i < discBeliefs.length; i++) {
            double hi = iv[i].boundary;
            System.out.printf(
                "\tP(%s in %g..%g)=%g\n", nodeId, lo, hi, discBeliefs[i]);
            lo = hi;
        }
    } else {
        double[] stats = net.getNodeSampleStats(nodeHandle);
        System.out.printf("%s: mean=%g stddev=%g min=%g max=%g\n",
            nodeId, stats[0], stats[1], stats[2], stats[3]);
    }
}

static void setUniformIntervals(Network net, int nodeHandle, int count) {
    double[] bounds = net.getNodeEquationBounds(nodeHandle);
    double lo = bounds[0];
    double hi = bounds[1];

    DiscretizationInterval[] iv = new DiscretizationInterval[count];
    for (int i = 0; i < count; i++) {
        iv[i] = new DiscretizationInterval(
            null, lo + (i + 1) * (hi - lo) / count);
    }

    net.setNodeEquationDiscretization(nodeHandle, iv);
}

static void updateAndShowStats(Network net) {
    net.updateBeliefs();
    for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h))
    {
        showStats(net, h);
    }
    System.out.println();
}
}

```

7.7.3 Tutorial7.cs

```

using System;
using Smile;

// Tutorial7 creates a network with three equation-based nodes
// performs the inference, then saves the model to disk.

namespace SmileNetTutorial
{
    class Tutorial7
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial7...");
            Network net = new Network();

```

```

net.OutlierRejectionEnabled = true;

CreateEquationNode(net,
    "tra", "Return Air Temperature",
    "tra=24", 23.9, 24.1,
    280, 100);

CreateEquationNode(net,
    "u_d", "Damper Control Signal",
    "u_d = Bernoulli(0.539)*0.8 + 0.2", 0, 1,
    160, 100);

int toa = CreateEquationNode(net,
    "toa", "Outside Air Temperature",
    "toa=Normal(11,15)", -10, 40,
    60, 100);

// tra, toa and u_d are referenced in equation
// arcs are created automatically
int tma = CreateEquationNode(net,
    "tma", "Mixed Air Temperature",
    "tma=toa*u_d+(tra-tra*u_d)", 10, 30,
    110, 200);

SetUniformIntervals(net, toa, 5);
SetUniformIntervals(net, tma, 4);

Console.WriteLine("Results with no evidence:");
UpdateAndShowStats(net);

net.SetContEvidence(toa, 28.5);
Console.WriteLine("Results with outside air temperature set to 28.5:");
UpdateAndShowStats(net);

net.ClearEvidence(toa);
Console.WriteLine("Results with mixed air temperature set to 21:");
net.SetContEvidence(tma, 21.0);
UpdateAndShowStats(net);

net.WriteFile("tutorial7.xdsl");
Console.WriteLine("Tutorial7 complete: Network written to tutorial7.xdsl");
}

private static int CreateEquationNode(
    Network net, String id, String name,
    String equation, double loBound, double hiBound,
    int xPos, int yPos)
{
    int handle = net.AddNode(Network.NodeType.Equation, id);
    net.SetNodeName(handle, name);
    net.SetNodePosition(handle, xPos, yPos, 85, 55);
    net.SetNodeEquation(handle, equation);
    net.SetNodeEquationBounds(handle, loBound, hiBound);
    return handle;
}

private static void ShowStats(Network net, int nodeHandle)
{
    String nodeId = net.GetNodeId(nodeHandle);

    if (net.IsEvidence(nodeHandle))

```

```

    {
        double v = net.GetContEvidence(nodeHandle);
        Console.WriteLine("{0} has evidence set ({1})", nodeId, v);
        return;
    }

    if (net.IsValueDiscretized(nodeHandle))
    {
        Console.WriteLine("{0} is discretized.", nodeId);
        DiscretizationInterval[] iv = net.GetNodeEquationDiscretization(nodeHandle);
        double[] bounds = net.GetNodeEquationBounds(nodeHandle);
        double[] discBeliefs = net.GetNodeValue(nodeHandle);
        double lo = bounds[0];
        for (int i = 0; i < discBeliefs.Length; i++)
        {
            double hi = iv[i].Boundary;
            Console.WriteLine("\tP({0} in {1}..{2})={3}",
                nodeId, lo, hi, discBeliefs[i]);
            lo = hi;
        }
    }
    else
    {
        double[] stats = net.GetNodeSampleStats(nodeHandle);
        Console.WriteLine("{0}: mean={1} stddev={2} min={3} max={4}",
            nodeId, stats[0], stats[1], stats[2], stats[3]);
    }
}

private static void SetUniformIntervals(
    Network net, int nodeHandle, int count)
{
    double[] bounds = net.GetNodeEquationBounds(nodeHandle);
    double lo = bounds[0];
    double hi = bounds[1];

    DiscretizationInterval[] iv = new DiscretizationInterval[count];
    for (int i = 0; i < count; i++)
    {
        iv[i] = new DiscretizationInterval(null, lo + (i + 1) * (hi - lo) / count);
    }

    net.SetNodeEquationDiscretization(nodeHandle, iv);
}

private static void UpdateAndShowStats(Network net)
{
    net.UpdateBeliefs();
    for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
    {
        ShowStats(net, h);
    }
    Console.WriteLine();
}
}
}

```

7.7.4 Tutorial7.R

```

library(rSMILE)
source("License.R")

# Tutorial7 creates a network with three equation-based nodes
# performs the inference, then saves the model to disk.

createEquationNode = function(net, id, name, equation, loBound,
                              hiBound, xPos, yPos) {
  handle <- net$addNode(net$NodeType$EQUATION, id)
  net$setNodeName(handle, name)
  net$setNodeEquation(handle, equation)
  net$setNodeEquationBounds(handle, loBound, hiBound)
  net$setNodePosition(handle, xPos, yPos, 85, 55)

  return(handle)
}

showStats = function(net, nodeHandle) {
  nodeId <- net$getNodeId(nodeHandle)
  if (net$isEvidence(nodeHandle)) {
    v <- net$getContentEvidence(nodeHandle)
    cat(sprintf("%s has evidence set (%g)\n", nodeId, v))
    return()
  }

  if (net$isValueDiscretized(nodeHandle)) {
    cat(sprintf("%s is discretized.\n", nodeId))
    iv <- net$getNodeEquationDiscretization(nodeHandle)
    bounds <- net$getNodeEquationBounds(nodeHandle)
    discBeliefs <- net$getNodeValue(nodeHandle)
    lo <- bounds[1]
    for(i in 0:(length(discBeliefs)-1)) {
      hi <- iv[[i+1]]$boundary
      cat(sprintf("\tP(%s in %g..%g)=%g\n",
                  nodeId, lo, hi, discBeliefs[i+1]))
      lo <- hi
    }
  } else {
    stats <- net$getNodeSampleStats(nodeHandle)
    cat(sprintf("%s: mean=%g stddev=%g min=%g max=%g\n",
                nodeId, stats[1], stats[2], stats[3], stats[4]))
  }
}

setUniformIntervals = function(net, nodeHandle, count) {
  bounds <- net$getNodeEquationBounds(nodeHandle)
  lo <- bounds[1]
  hi <- bounds[2]

  iv <- lapply(rep("DiscretizationInterval", count), new)
  num <- 1
  for(i in iv) {
    i$id <- ""
    i$boundary <- lo + num * (hi - lo) / count
    num <- num + 1
  }
  net$setNodeEquationDiscretization(nodeHandle, iv)
}

```

```

updateAndShowStats = function(net) {
  net$updateBeliefs()
  nodes <- net$getAllNodes()
  for (h in nodes) {
    showStats(net, h)
  }
  cat("\n")
}

cat("Starting Tutorial7...\n")
net <- Network()

net$setOutlierRejectionEnabled(TRUE)

createEquationNode(net, "tra", "Return Air Temperature",
  "tra=24", 23.9, 24.1, 280, 100)

createEquationNode(net, "u_d", "Damper Control Signal",
  "u_d = Bernoulli(0.539)*0.8 + 0.2", 0, 1, 160, 100)

toa <- createEquationNode(net, "toa", "Outside Air Temperature",
  "toa=Normal(11,15)", -10, 40, 60, 100)

# tra, toa and u_d are referenced in equation
# arcs are created automatically
tma <- createEquationNode(net, "tma", "Mixed Air Temperature",
  "tma=toa*u_d+(tra-tra*u_d)", 10, 30, 110, 200)

setUniformIntervals(net, toa, 5)
setUniformIntervals(net, tma, 4)

cat("Results with no evidence:\n")
updateAndShowStats(net)

net$setContEvidence(toa, 28.5)
cat("Results with outside air temperature set to 28.5:\n")
updateAndShowStats(net)

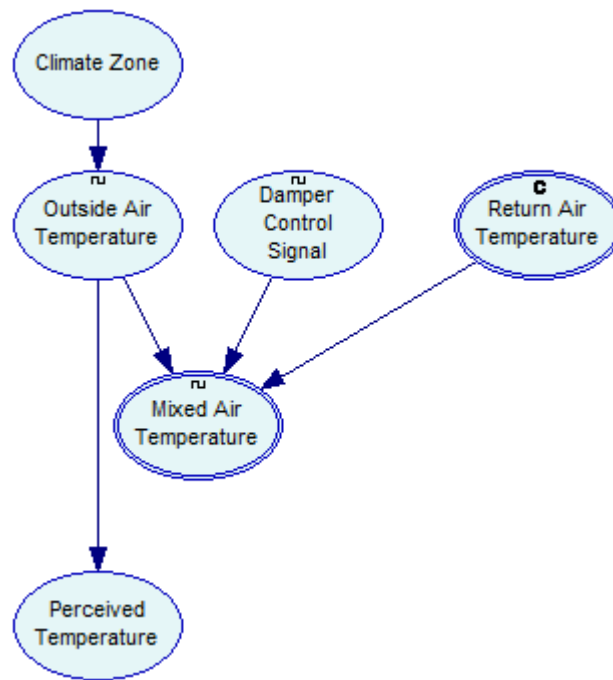
net$clearEvidence(toa)
cat("Results with mixed air temperature set to 21:\n")
net$setContEvidence(tma, 21.0)
updateAndShowStats(net)

net$writeFile("tutorial7.xdsl")
cat("Tutorial7 complete: Network written to tutorial7.xdsl\n")

```

7.8 Tutorial 8: Hybrid model

We extend the model described in [Tutorial 7](#)¹³² by adding two discrete nodes: *Climate Zone* and *Perceived Temperature*. *Climate Zone* refines the probability distribution of the outside air temperature and *Perceived Temperature* is an additional input originating from a subjective perception of the temperature, useful in case of a failure in the outside temperature sensor.



After loading the network created in the previous tutorial, the program adds two discrete nodes using the `CreateCptNode` helper function first seen in [Tutorial 1](#)⁸⁴. The arc from *Climate Zone* (node identifier is *zone*) to *Outside Air Temperature* (node identifier is *toa*) is created by changing the equation of the latter:

Python:

```
net.set_node_equation("toa", 'toa=If(zone="Desert",Normal(22,5),Normal(11,10))')
```

Java:

```
net.setNodeEquation(toaHandle, "toa=If(zone=\"Desert\",Normal(22,5),Normal(11,10))");
```

C#:

```
net.SetNodeEquation(toaHandle, "toa=If(zone=\"Desert\",Normal(22,5),Normal(11,10))");
```

R:

```
net$setNodeEquation(toaHandle, "toa=If(zone=\"Desert\",Normal(22,5),Normal(11,10))")
```

In the Java and C# code, we need to escape the double quote characters in order to use the text literal representing the *Desert* outcome of the parent. Python allows for single quotes to be used as string literal delimiter.

The new equation switches between two normal distribution based on the outcome of the parent. The equation could be also written in the following ways, all being functionally identical (assuming that *Climate Zone* has two outcomes, *Temperate* and *Desert*):

```
toa=zone="Desert" ? Normal(22,5) : Normal(11,10)
toa=zone="Temperate" ? Normal(11,10) : Normal(22,5)
toa=Switch(zone, "Desert",Normal(22,5),"Temperate",Normal(11,10))
toa=Choose(zone,Normal(11,10),Normal(22,5))
```

To create the arc from *Outside Air Temperature* to *Perceived Temperature* the program calls `Network.addArc`. The model loaded from disk had 5 discretization intervals already defined for *Outside Air Temperature*. With 5 possible discretized outcomes of its single parent and its own 3 outcomes, the CPT for *Perceived Temperature* has $3 \times 5 = 15$ entries. It is initialized by a call to `Network.setNodeDefinition` with an appropriately sized array as input. Note that for simplicity we do not change the default uniform distribution for the binary *Climate Zone* node.

The program performs inference for each of the *Climate Zone* outcomes set as evidence. The output displayed for the *Outside Air Temperature* node (*toa*) shows changing mean and standard deviation. Finally, the network is saved to disk. This concludes the tutorial.

7.8.1 Tutorial8.py

```
import pysmile

# Tutorial8 loads continuous model from the XDSL file written by Tutorial7,
# then adds discrete nodes to create a hybrid model. Inference is performed
# and model is saved to disk.

class Tutorial8:
    def __init__(self):
        print("Starting Tutorial8...")
        net = pysmile.Network()

        net.read_file("tutorial7.xdsl")

        self.create_cpt_node(net,
            "zone", "Climate Zone",
            ["Temperate", "Desert" ],
            60, 20)
        toa_handle = net.get_node("toa")
        net.set_node_equation("toa", 'toa=If(zone="Desert",Normal(22,5),Normal(11,10))')

        perceived_handle = self.create_cpt_node(net,
            "perceived", "Perceived Temperature",
            ["Hot", "Warm", "Cold"],
            60, 300)

        net.add_arc(toa_handle, perceived_handle)

        perceived_probs = [
            0,      # P(perceived=Hot | toa in -10..0)
            0.02,   # P(perceived=Warm|toa in -10..0)
            0.98,   # P(perceived=Cold|toa in -10..0)
            0.05,   # P(perceived=Hot | toa in 0..10)
            0.15,   # P(perceived=Warm|toa in 0..10)
            0.80,   # P(perceived=Cold|toa in 0..10)
            0.10,   # P(perceived=Hot | toa in 10..20)
            0.80,   # P(perceived=Warm|toa in 10..20)
            0.10,   # P(perceived=Cold|toa in 10..20)
            0.80,   # P(perceived=Hot | toa in 20..30)
            0.15,   # P(perceived=Warm|toa in 20..30)
            0.05,   # P(perceived=Cold|toa in 20..30)
            0.98,   # P(perceived=Hot | toa in 30..40)
            0.02,   # P(perceived=Warm|toa in 30..40)
            0       # P(perceived=Cold|toa in 30..40)
        ]

        net.set_node_definition(perceived_handle, perceived_probs)

        net.set_evidence("zone", "Temperate")
        print("Results in temperate zone:")
        self.update_and_show_stats(net)

        net.set_evidence("zone", "Desert")
```

```

print("Results in desert zone:")
self.update_and_show_stats(net)

net.write_file("tutorial8.xdsl")
print("Tutorial8 complete: Network written to tutorial8.xdsl")

def show_stats(self, net, node_handle):
    node_id = net.get_node_id(node_handle)

    if net.is_evidence(node_handle):
        v = net.get_cont_evidence(node_handle)
        print(f"{node_id} has evidence set: {v}")
        return

    if net.is_value_discretized(node_handle):
        print(node_id + " is discretized.")
        iv = net.get_node_equation_discretization(node_handle)
        bounds = net.get_node_equation_bounds(node_handle)
        disc_beliefs = net.get_node_value(node_handle)
        lo = bounds[0]
        for i in range(0, len(disc_beliefs)):
            hi = iv[i].boundary
            print(f"\tP({node_id} in {lo}..{hi})={disc_beliefs[i]}")
            lo = hi
    else:
        stats = net.get_node_sample_stats(node_handle)
        print(f"{node_id}: mean={stats[0]} stddev={stats[1]} min={stats[2]} max={stats[3]}")

def update_and_show_stats(self, net):
    net.update_beliefs()
    for h in net.get_all_nodes():
        if net.get_node_type(h) == pysmile.NodeType.EQUATION:
            self.show_stats(net, h)
    print()

def create_cpt_node(self, net, id, name, outcomes, x_pos, y_pos):
    handle = net.add_node(pysmile.NodeType.CPT, id)

    net.set_node_name(handle, name)
    net.set_node_position(handle, x_pos, y_pos, 85, 55)

    initial_outcome_count = net.get_outcome_count(handle)

    for i in range(0, initial_outcome_count):
        net.set_outcome_id(handle, i, outcomes[i])

    for i in range(initial_outcome_count, len(outcomes)):
        net.add_outcome(handle, outcomes[i])

    return handle

```

7.8.2 Tutorial8.java

```

package tutorials;

import smile.*;

// Tutorial8 loads continuous model from the XDSL file written by Tutorial7,
// then adds discrete nodes to create a hybrid model. Inference is performed
// and model is saved to disk.

```

```

public class Tutorial8 {
    public static void run() {
        System.out.println("Starting Tutorial8...");
        Network net = new Network();

        net.readFile("tutorial7.xdsl");

        createCptNode(
            net, "zone", "Climate Zone",
            new String[] { "Temperate", "Desert" },
            60, 20);

        int toaHandle = net.getNode("toa");
        net.setNodeEquation(toaHandle, "toa=If(zone=\"Desert\",Normal(22,5),Normal(11,10))");

        int perceivedHandle = createCptNode(
            net, "perceived", "Perceived Temperature",
            new String[] { "Hot", "Warm", "Cold" },
            60, 300);
        net.addArc(toaHandle, perceivedHandle);

        double[] perceivedProbs = new double[] {
            0, // P(perceived=Hot |toa in -10..0)
            0.02, // P(perceived=Warm|toa in -10..0)
            0.98, // P(perceived=Cold|toa in -10..0)
            0.05, // P(perceived=Hot |toa in 0..10)
            0.15, // P(perceived=Warm|toa in 0..10)
            0.80, // P(perceived=Cold|toa in 0..10)
            0.10, // P(perceived=Hot |toa in 10..20)
            0.80, // P(perceived=Warm|toa in 10..20)
            0.10, // P(perceived=Cold|toa in 10..20)
            0.80, // P(perceived=Hot |toa in 20..30)
            0.15, // P(perceived=Warm|toa in 20..30)
            0.05, // P(perceived=Cold|toa in 20..30)
            0.98, // P(perceived=Hot |toa in 30..40)
            0.02, // P(perceived=Warm|toa in 30..40)
            0 // P(perceived=Cold|toa in 30..40)
        };
        net.setNodeDefinition(perceivedHandle, perceivedProbs);

        net.setEvidence("zone", "Temperate");
        System.out.println("Results in temperate zone:");
        updateAndShowStats(net);

        net.setEvidence("zone", "Desert");
        System.out.println("Results in desert zone:\n");
        updateAndShowStats(net);

        net.writeFile("tutorial8.xdsl");
        System.out.println("Tutorial8 complete: Network written to tutorial8.xdsl");
    }

    static void showStats(Network net, int nodeHandle) {
        String nodeId = net.getNodeId(nodeHandle);

        if (net.isEvidence(nodeHandle)) {
            double v = net.getContEvidence(nodeHandle);
            System.out.printf("%s has evidence set (%g)\n", nodeId, v);
            return;
        }
    }
}

```

```

        if (net.isValueDiscretized(nodeHandle)) {
            System.out.printf("%s is discretized.\n", nodeId);
            DiscretizationInterval[] iv =
                net.getNodeEquationDiscretization(nodeHandle);
            double[] bounds = net.getNodeEquationBounds(nodeHandle);
            double[] discBeliefs = net.getNodeValue(nodeHandle);
            double lo = bounds[0];
            for (int i = 0; i < discBeliefs.length; i++) {
                double hi = iv[i].boundary;
                System.out.printf(
                    "\tP(%s in %g..%g)=%g\n", nodeId, lo, hi, discBeliefs[i]);
                lo = hi;
            }
        } else {
            double[] stats = net.getNodeSampleStats(nodeHandle);
            System.out.printf("%s: mean=%g stddev=%g min=%g max=%g\n",
                nodeId, stats[0], stats[1], stats[2], stats[3]);
        }
    }

    static void updateAndShowStats(Network net) {
        net.updateBeliefs();
        for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h))
        {
            if (net.getNodeType(h) == Network.NodeType.EQUATION) {
                showStats(net, h);
            }
        }
        System.out.println();
    }

    private static int createCptNode(
        Network net, String id, String name,
        String[] outcomes, int xPos, int yPos) {
        int handle = net.addNode(Network.NodeType.CPT, id);

        net.setNodeName(handle, name);
        net.setNodePosition(handle, xPos, yPos, 85, 55);

        int initialOutcomeCount = net.getOutcomeCount(handle);
        for (int i = 0; i < initialOutcomeCount; i++) {
            net.setOutcomeId(handle, i, outcomes[i]);
        }

        for (int i = initialOutcomeCount; i < outcomes.length; i++) {
            net.addOutcome(handle, outcomes[i]);
        }

        return handle;
    }
}

```

7.8.3 Tutorial8.cs

```

using System;
using Smile;

// Tutorial8 loads continuous model from the XDSL file written by Tutorial7,
// then adds discrete nodes to create a hybrid model. Inference is performed

```

```
// and model is saved to disk.

namespace SmileNetTutorial
{
    class Tutorial8
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial8...");
            Network net = new Network();

            net.ReadFile("tutorial7.xdsl");

            CreateCptNode(
                net, "zone", "Climate Zone",
                new String[] { "Temperate", "Desert" },
                60, 20);

            int toaHandle = net.GetNode("toa");
            net.SetNodeEquation(toaHandle,
                "toa=If(zone=\"Desert\",Normal(22,5),Normal(11,10))");

            int perceivedHandle = CreateCptNode(
                net, "perceived", "Perceived Temperature",
                new String[] { "Hot", "Warm", "Cold" },
                60, 300);
            net.AddArc(toaHandle, perceivedHandle);

            double[] perceivedProbs = new double[] {
                0, // P(perceived=Hot | toa in -10..0)
                0.02, // P(perceived=Warm | toa in -10..0)
                0.98, // P(perceived=Cold | toa in -10..0)
                0.05, // P(perceived=Hot | toa in 0..10)
                0.15, // P(perceived=Warm | toa in 0..10)
                0.80, // P(perceived=Cold | toa in 0..10)
                0.10, // P(perceived=Hot | toa in 10..20)
                0.80, // P(perceived=Warm | toa in 10..20)
                0.10, // P(perceived=Cold | toa in 10..20)
                0.80, // P(perceived=Hot | toa in 20..30)
                0.15, // P(perceived=Warm | toa in 20..30)
                0.05, // P(perceived=Cold | toa in 20..30)
                0.98, // P(perceived=Hot | toa in 30..40)
                0.02, // P(perceived=Warm | toa in 30..40)
                0 // P(perceived=Cold | toa in 30..40)
            };
            net.SetNodeDefinition(perceivedHandle, perceivedProbs);

            net.SetEvidence("zone", "Temperate");
            Console.WriteLine("Results in temperate zone:");
            UpdateAndShowStats(net);

            net.SetEvidence("zone", "Desert");
            Console.WriteLine("Results in desert zone:\n");
            UpdateAndShowStats(net);

            net.WriteFile("tutorial8.xdsl");
            Console.WriteLine("Tutorial8 complete: Network written to tutorial8.xdsl");
        }
    }

    private static int CreateEquationNode(
        Network net, String id, String name,
```

```

        String equation, double loBound, double hiBound,
        int xPos, int yPos)
    {
        int handle = net.AddNode(Network.NodeType.Equation, id);
        net.SetNodeName(handle, name);
        net.SetNodePosition(handle, xPos, yPos, 85, 55);
        net.SetNodeEquation(handle, equation);
        net.SetNodeEquationBounds(handle, loBound, hiBound);
        return handle;
    }

private static void ShowStats(Network net, int nodeHandle)
{
    String nodeId = net.GetNodeId(nodeHandle);

    if (net.IsEvidence(nodeHandle))
    {
        double v = net.GetContEvidence(nodeHandle);
        Console.WriteLine("{0} has evidence set ({1})", nodeId, v);
        return;
    }

    if (net.IsValueDiscretized(nodeHandle))
    {
        Console.WriteLine("{0} is discretized.", nodeId);
        DiscretizationInterval[] iv =
            net.GetNodeEquationDiscretization(nodeHandle);
        double[] bounds = net.GetNodeEquationBounds(nodeHandle);
        double[] discBeliefs = net.GetNodeValue(nodeHandle);
        double lo = bounds[0];
        for (int i = 0; i < discBeliefs.Length; i++)
        {
            double hi = iv[i].Boundary;
            Console.WriteLine("\tP({0} in {1}..{2})={3}",
                nodeId, lo, hi, discBeliefs[i]);
            lo = hi;
        }
    }
    else
    {
        double[] stats = net.GetNodeSampleStats(nodeHandle);
        Console.WriteLine("{0}: mean={1} stddev={2} min={3} max={4}",
            nodeId, stats[0], stats[1], stats[2], stats[3]);
    }
}

private static void UpdateAndShowStats(Network net)
{
    net.UpdateBeliefs();
    for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
    {
        if (net.GetNodeType(h) == (int)Network.NodeType.Equation)
        {
            ShowStats(net, h);
        }
    }
    Console.WriteLine();
}

private static int CreateCptNode(
    Network net, String id, String name,

```

```

    String[] outcomes, int xPos, int yPos)
    {
        int handle = net.AddNode(Network.NodeType.Cpt, id);

        net.SetNodeName(handle, name);
        net.SetNodePosition(handle, xPos, yPos, 85, 55);

        int initialOutcomeCount = net.GetOutcomeCount(handle);
        for (int i = 0; i < initialOutcomeCount; i++)
        {
            net.SetOutcomeId(handle, i, outcomes[i]);
        }

        for (int i = initialOutcomeCount; i < outcomes.Length; i++)
        {
            net.AddOutcome(handle, outcomes[i]);
        }

        return handle;
    }
}

```

7.8.4 Tutorial8.R

```

library(rSMILE)
source("License.R")

```

```

# Tutorial8 loads continuous model from the XDSL file written by Tutorial7,
# then adds discrete nodes to create a hybrid model. Inference is performed
# and model is saved to disk.

```

```

createCptNode = function(net, id, name, outcomes, xPos, yPos) {
    handle <- net$addNode(18, id)
    initialOutcomeCount <- net$getOutcomeCount(handle)
    count <- length(outcomes)
    net$setNodeName(handle, name)
    net$setNodePosition(handle, xPos, yPos, 85L, 55L)
    if (!is.null(outcomes)) {
        sapply(0:(initialOutcomeCount-1),
            function(x) net$setOutcomeId(handle, x, outcomes[x+1]))
        if (initialOutcomeCount < count) {
            sapply(initialOutcomeCount:(count-1),
                function(x) net$addOutcome(handle, outcomes[x+1]))
        }
    }

    return(handle)
}

showStats = function(net, nodeHandle) {
    nodeId <- net$getNodeId(nodeHandle)
    if (net$isEvidence(nodeHandle)) {
        v <- net$getContEvidence(nodeHandle)
        cat(sprintf("%s has evidence set (%g)\n", nodeId, v))
        return()
    }

    if (net$isValueDiscretized(nodeHandle)) {
        cat(sprintf("%s is discretized.\n", nodeId))
    }
}

```

```

        iv <- net$getNodeEquationDiscretization(nodeHandle)
        bounds <- net$getNodeEquationBounds(nodeHandle)
        discBeliefs <- net$getNodeValue(nodeHandle)
        lo <- bounds[1]
        for(i in 0:(length(discBeliefs)-1)) {
            hi <- iv[[i+1]]$boundary
            cat(sprintf("\tP(%s in %g..%g)=%g\n",
                nodeId, lo, hi, discBeliefs[i+1]))
            lo <- hi
        }
    } else {
        stats <- net$getNodeSampleStats(nodeHandle)
        cat(sprintf("%s: mean=%g stddev=%g min=%g max=%g\n",
            nodeId, stats[1], stats[2], stats[3], stats[4]))
    }
}

updateAndShowStats = function(net) {
    net$updateBeliefs()
    nodes <- net$getAllNodes()
    for (h in nodes) {
        if(net$getNodeType(h) == net$NodeType$EQUATION) {
            showStats(net, h)
        }
    }
    cat("\n")
}

cat("Starting Tutorial8...\n")
net <- Network()

net$readFile("tutorial7.xdsl")

createCptNode(net, "zone", "Climate Zone", c("Temperate", "Desert"), 60, 20)

toaHandle <- net$getNode("toa")
net$setNodeEquation(toaHandle, "toa=If(zone=\"Desert\",Normal(22,5),Normal(11,10))")

perceivedHandle <- createCptNode(net, "perceived", "Perceived Temperature",
    c("Hot", "Warm", "Cold"), 60, 300)

net$addArc(toaHandle, perceivedHandle)

perceivedProbs = c( 0.00,
                    0.02,
                    0.98,
                    0.05,
                    0.15,
                    0.80,
                    0.10,
                    0.80,
                    0.10,
                    0.80,
                    0.15,
                    0.05,
                    0.98,
                    0.02,
                    0.00 )

net$setNodeDefinition(perceivedHandle, perceivedProbs)

```

```

net$setEvidence("zone", "Temperate")
cat("Results in temperate zone:\n")
updateAndShowStats(net)

net$setEvidence("zone", "Desert")
cat("Results in desert zone:\n")
updateAndShowStats(net)

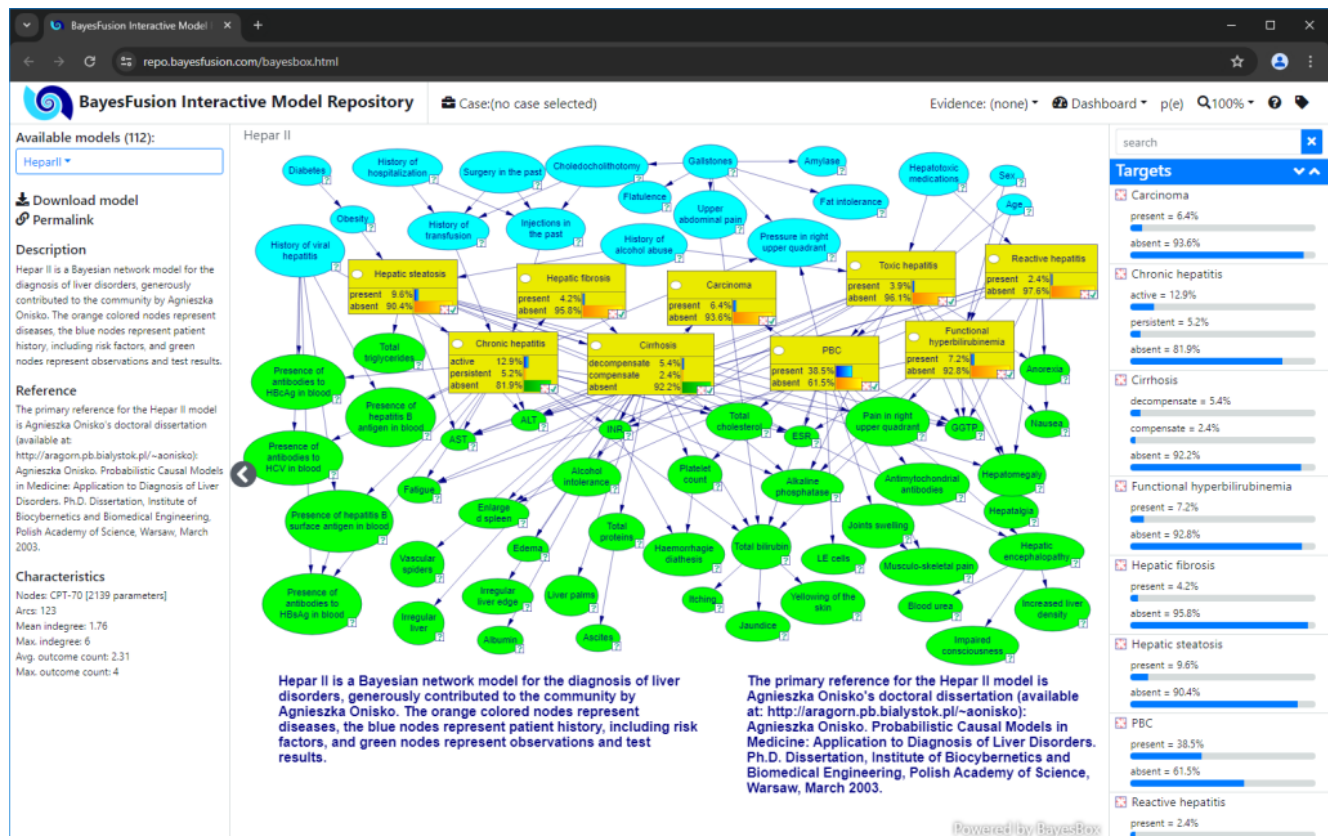
net$writeFile("tutorial8.xdsl")

cat("Tutorial8 complete: Network written to tutorial8.xdsl\n")

```

7.9 Tutorial 9: Diagnosis

The program in this section uses one of the example discrete networks distributed in GeNIe, which can be downloaded directly using the following link: <https://support.bayesfusion.com/docs/Examples/Discrete%20Bayesian%20Networks/HeparII.xdsl>. The model is also available in BayesFusion's online repository at <https://repo.bayesfusion.com/network/permalink?net=Small+BNs%2FHeparII.xdsl>.



HeparII.xdsl is a Bayesian network model for the diagnosis of liver disorders. It has diagnostic roles defined for its nodes. The model nodes were formatted to use yellow as a background for fault nodes, and green/blue for observation nodes. Note that while binary fault nodes have one fault state (*present*), there are also two fault nodes with three states. These nodes have two out of three states marked as faulty, like *active* and *persistent* in the case of *Chronic Hepatitis*.

Our program loads the network at startup and calls its helper function `printDiagTypes` to output the information about diagnostic roles. The first lines of the output look like this:

Node Id	Diagnostic Type	Fault Outcomes
alcoholism	NodeDiagType.OBSERVATION	n/a
vh_amn	NodeDiagType.OBSERVATION	n/a
hepatotoxic	NodeDiagType.OBSERVATION	n/a
THepatitis	NodeDiagType.FAULT	['present']
hospital	NodeDiagType.OBSERVATION	n/a
surgery	NodeDiagType.OBSERVATION	n/a
gallstones	NodeDiagType.OBSERVATION	n/a
choledocholithotomy	NodeDiagType.OBSERVATION	n/a
injections	NodeDiagType.OBSERVATION	n/a
transfusion	NodeDiagType.OBSERVATION	n/a
ChHepatitis	NodeDiagType.FAULT	['active', 'persistent']
sex	NodeDiagType.OBSERVATION	n/a
...		

The `printDiagTypes` function iterates over the nodes in the network. Its main loop calls `Network.getNodeDiagType` to determine the diagnostic role. If the node is a fault, its outcomes are also inspected using `Network.isFaultOutcome`.

Python:

```
diag_type = net.get_node_diag_type(node_id)
fault_outcomes = []
if diag_type == pysmile.NodeDiagType.FAULT:
    for outcome_id in net.get_outcome_ids(node_id):
        if net.is_fault_outcome(node_id, outcome_id):
            fault_outcomes.append(outcome_id)
```

Java:

```
int diagType = net.getNodeDiagType(nodeId);
List<String> faultOutcomes = new ArrayList<>();
if (diagType == Network.NodeDiagType.FAULT) {
    for (String outcomeId : net.getOutcomeIds(nodeId)) {
        if (net.isFaultOutcome(nodeId, outcomeId)) {
            faultOutcomes.add(outcomeId);
        }
    }
}
```

C#:

```
Network.NodeDiagType diagType = net.GetNodeDiagType(nodeId);
List<string> faultOutcomes = new List<string>();
if (diagType == Network.NodeDiagType.Fault)
{
    foreach (string outcomeId in net.GetOutcomeIds(nodeId))
    {
        if (net.IsFaultOutcome(nodeId, outcomeId))
        {
            faultOutcomes.Add(outcomeId);
        }
    }
}
```

R:

```
diagType <- net$getNodeDiagType(nodeId)
faultOutcomes <- character()
if (diagType == net$NodeDiagType$FAULT) {
    outcomeIds <- net$getOutcomeIds(nodeId)
    for (outcomeId in outcomeIds) {
        if (net$isFaultOutcome(nodeId, outcomeId)) {
            faultOutcomes <- c(faultOutcomes, outcomeId)
        }
    }
}
```

We now proceed to the actual diagnosis: the `DiagNetwork` object is instantiated. In its constructor, the fault probabilities are calculated and the most likely fault is selected to be pursued. `DiagNetwork` uses zero-based integer indices to identify faults (node/node outcome pairs). The `printFaultIndices` helper function iterates over faults. It uses `DiagNetwork.getFaultCount`, `getFaultNodeId` and `getFaultOutcomeId` to emit the following output:

```
Diagnostic network has 11 faults (node/outcome pairs)
Fault Index  Fault Node Id      Fault Outcome Id
0            THepatitis        present
1            ChHepatitis       active
2            ChHepatitis       persistent
3            PBC               present
4            fibrosis          present
5            Steatosis         present
6            Cirrhosis         decompensate
7            Cirrhosis         compensate
8            Hyperbilirubinemia present
9            RHepatitis        present
10           carcinoma         present
```

We can read the index of the currently pursued fault with `DiagNetwork.getPursuedFault`. Initially, the most likely fault is at index 3, which represents the node *PBC* and its outcome *present*.

Python:

```
pursued_fault_idx = diag.get_pursued_fault()
print(f"The default (most likely) pursued fault is at index {pursued_fault_idx}: "
      + f"{diag.get_fault_node_id(pursued_fault_idx)}"
      + f"={diag.get_fault_outcome_id(pursued_fault_idx)}")
```

Java:

```
int pursuedFaultIdx = diag.getPursuedFault();
System.out.printf("The default (most likely) pursued fault is at index %d: %s=%s\n",
    pursuedFaultIdx, diag.getFaultNodeId(pursuedFaultIdx),
    diag.getFaultOutcomeId(pursuedFaultIdx));
```

C#:

```
int pursuedFaultIdx = diag.GetPursuedFault();
Console.WriteLine("The default (most likely) pursued fault is at index {0}: {1}={2}",
    pursuedFaultIdx,
    diag.GetFaultNodeId(pursuedFaultIdx), diag.GetFaultOutcomeId(pursuedFaultIdx));
```

R:

```
pursuedFaultIdx <- diag$getPursuedFault()
cat(sprintf("The default (most likely) pursued fault is at index %d: %s=%s\n",
    pursuedFaultIdx,
    diag$getFaultNodeId(pursuedFaultIdx), diag$getFaultOutcomeId(pursuedFaultIdx)))
```

The program performs the diagnostic update without any observations instantiated. `DiagNetwork.update` is called, and results are returned as a `DiagResults` object. The diagnostic results contain two arrays, one for faults, and another for observations. The arrays are sorted: faults are sorted by probability with most likely faults at the top, and observations are sorted by the magnitude of the information gain. Our `printDiagResults` iterates over the elements of these arrays. `printFaultInfo` and `printObservationInfo` are used to access the members of the `FaultInfo` and `ObservationInfo` objects, respectively. The first lines of the fault/observation output look like this:

```
Faults:
Node Id      Outcome Id      Probability      Is Pursued
PBC          present        0.41341570632059377  Yes
ChHepatitis  active         0.1290048962276199   No
Steatosis    present        0.09585221239536176  No
carcinoma    present        0.0665709760500613   No
...
Observations:
```

Node Id	Measure	Cost	InfoGain
ama	0.5576308476489384	0.0	0.5576308476489384
cholesterol	0.4868445142223191	0.0	0.4868445142223191
ESR	0.42960779128083926	0.0	0.42960779128083926
bilirubin	0.41580785350162774	0.0	0.41580785350162774
age	-0.33803891617697684	0.0	-0.33803891617697684
skin	0.3100428612068602	0.0	0.3100428612068602
...			

The default diagnostic measure algorithm for single fault diagnosis uses the maximum magnitude of probability change in the pursued fault over all outcomes of the observation node. From the output, we can deduce that the *ama* node has an outcome that causes the probability $PBC=present$ to increase by 0.55763. Note also that *age* infoGain is negative due to (signed) probability change used as a diagnostic measure algorithm. For brevity, the tutorial program does not use `ObservationInfo.outcomeGain` array, which contains per-outcome infoGain. From `outcomeGain`, we could determine which outcome of the *ama* node is responsible for the maximum change in PBC . Other members of `ObservationInfo` not displayed by this tutorial are `faultPosteriors` and `observationPriors`. The `faultPosteriors` array contains probabilities of pursued fault after instantiating each outcome of the observation. The `observationPriors` array contains prior probabilities for each observation outcome.

We will now instantiate (observe) some observation nodes. `DiagNetwork.instantiateObservation` calls `Network.setEvidence` internally, and updates `DiagNetwork`'s state. **Always call `DiagNetwork.instantiateObservation` when the diagnostic session is in progress.** Calling `Network.setEvidence` directly may lead to inconsistent results or errors. Our observations are:

- *jaundice=present*
- *nausea=absent*
- *obesity=present*

The code is straightforward:

Python:

```
diag.instantiate_observation("jaundice", "present")
diag.instantiate_observation("nausea", "absent")
diag.instantiate_observation("obesity", "present")
```

Java:

```
diag.instantiateObservation("jaundice", "present");
diag.instantiateObservation("nausea", "absent");
diag.instantiateObservation("obesity", "present");
```

C#:

```
diag.InstantiateObservation("jaundice", "present");
diag.InstantiateObservation("nausea", "absent");
diag.InstantiateObservation("obesity", "present");
```

R:

```
diag$instantiateObservation("jaundice", "present")
diag$instantiateObservation("nausea", "absent")
diag$instantiateObservation("obesity", "present")
```

After another call to `DiagNetwork.update` and `printDiagResults`, we get the following output:

Faults:			
Node Id	Outcome Id	Probability	Is Pursued
PBC	present	0.3716679804854865	Yes
Steatosis	present	0.20822898960092975	No
ChHepatitis	active	0.12911036437662063	No
Cirrhosis	decompensate	0.08774271003265043	No
...			

```

Observations:
Node Id      Measure      Cost      InfoGain
ama          0.5940244757037065  0.0      0.5940244757037065
cholesterol  0.4557094807486714  0.0      0.4557094807486714
bilirubin    0.43194679397989816  0.0      0.43194679397989816
ESR          0.41696920311719615  0.0      0.41696920311719615
age          -0.3062170075414388  0.0      -0.3062170075414388
skin        0.2869860117172001  0.0      0.2869860117172001
...

```

With known *jaundice*, *nausea* and *obesity* the fault probabilities and observation infoGains have changed, but the *ama* node is still our most useful observation.

The next step is to change the outcome of *obesity* to *absent* and retract the observation of *nausea*. We will also switch our pursued fault to *Hyperbilirubinemia=present*.

Python:

```

diag.release_observation("nausea")
diag.instantiate_observation("obesity", "absent")
hyperbilirubinemia_idx = diag.get_fault_index("Hyperbilirubinemia", "present")
diag.set_pursued_fault(hyperbilirubinemia_idx)

```

Java:

```

diag.releaseObservation("nausea");
diag.instantiateObservation("obesity", "absent");
int hyperbilirubinemiaIdx = diag.getFaultIndex("Hyperbilirubinemia", "present");
diag.setPursuedFault(hyperbilirubinemiaIdx);

```

C#:

```

diag.ReleaseObservation("nausea");
diag.InstantiateObservation("obesity", "absent");
int hyperbilirubinemiaIdx = diag.GetFaultIndex("Hyperbilirubinemia", "present");
diag.SetPursuedFault(hyperbilirubinemiaIdx);

```

R:

```

diag$releaseObservation("nausea")
diag$instantiateObservation("obesity", "absent")
hyperbilirubinemiaIdx <- diag$getFaultIndex("Hyperbilirubinemia", "present")
diag$setPursuedFault(hyperbilirubinemiaIdx)

```

Another `DiagNetwork.update` and `printDiagResults` calls produce this output:

```

Node Id      Outcome Id      Probability      Is Pursued
PBC          present          0.3699731063429856  No
ChHepatitis  active           0.12884438248137664  No
Steatosis    present          0.08508721992530559  No
carcinoma    present          0.06083429718416463  No
ChHepatitis  persistent       0.05272028107403933  No
Hyperbilirubinemia  present          0.052337451916630945  Yes
...
Observations:
Node Id      Measure      Cost      InfoGain
age          0.2455407606076575  0.0      0.2455407606076575
hepatomegaly 0.04617967333309004  0.0      0.04617967333309004
bilirubin    0.042062826363683546  0.0      0.042062826363683546
ggtp        -0.03279259884194631  0.0      -0.03279259884194631
ESR          -0.024267251569646417  0.0      -0.024267251569646417
ama          -0.023463595723063817  0.0      -0.023463595723063817
...

```

We can immediately notice that the major factor determining the probability of *Hyperbilirubinemia=present* is *age*. Of course in the real-world scenario, we would know the age of the patient before starting the diagnosis.

Our final diagnosis with a single pursued fault will switch to a cross-entropy measure for infoGain. We do not change observations and the pursued faults remain the same.

Python:

```
diag.set_single_fault_algorithm(pysmile.SingleFaultAlgorithmType.CROSSENTROPY)
```

Java:

```
diag.setSingleFaultAlgorithm(DiagNetwork.SingleFaultAlgorithmType.CROSSENTROPY);
```

C#:

```
diag.SingleFaultAlgorithm = DiagNetwork.SingleFaultAlgorithmType.Crossentropy;
```

R:

```
diag$setSingleFaultAlgorithm(diag$SingleFaultAlgorithmType$CROSSENTROPY)
```

The output from `DiagNetwork.update` now has non-negative infoGain for all observations. Note that fault probabilities do not change when the diagnostic measure algorithm is changed, therefore the fragment of the tutorial output below omits the fault info.

Observations:			
Node Id	Measure	Cost	InfoGain
age	0.05465458670852352	0.0	0.05465458670852352
hepatomegaly	0.01315927809591283	0.0	0.01315927809591283
ggtp	0.010610204503726073	0.0	0.010610204503726073
bilirubin	0.006204569424644146	0.0	0.006204569424644146
sex	0.0031695792371849163	0.0	0.0031695792371849163
pain_ruq	0.00263781234873367	0.0	0.00263781234873367
ESR	0.0026302997853497323	0.0	0.0026302997853497323
ama	0.0025319015011320833	0.0	0.0025319015011320833
hepatalgia	0.001572547559233084	0.0	0.001572547559233084
cholesterol	9.548883878895409E-4	0.0	9.548883878895409E-4
...			

Without changing the observations, we now proceed to multi-fault diagnosis. We will keep *Hyperbilirubinemia=present* and add *Steatosis=present*. We know the fault index for the first fault already, but need to get the index for the second fault.

Python:

```
steatosis_idx = diag.get_fault_index("Steatosis", "present")
diag.set_pursued_faults([steatosis_idx, hyperbilirubinemia_idx])
```

Java:

```
int steatosisIdx = diag.getFaultIndex("Steatosis", "present");
diag.setPursuedFaults(new int[]{steatosisIdx, hyperbilirubinemiaIdx});
```

C#:

```
int steatosisIdx = diag.GetFaultIndex("Steatosis", "present");
diag.SetPursuedFaults(new int[] { steatosisIdx, hyperbilirubinemiaIdx });
```

R:

```
steatosisIdx <- diag$getFaultIndex("Steatosis", "present")
diag$setPursuedFaults(c(steatosisIdx, hyperbilirubinemiaIdx))
```

After a `DiagNetwork.update` we will get identical fault probabilities (because observations did not change), but the infoGains will be different:

Observations:			
Node Id	Measure	Cost	InfoGain
triglycerides	0.32730161296292315	0.0	0.32730161296292315
age	0.2455407606076575	0.0	0.2455407606076575
alcoholism	0.14812380191186358	0.0	0.14812380191186358
inr	0.13221077172053503	0.0	0.13221077172053503
irregular_liver	0.11230469436997036	0.0	0.11230469436997036
spleen	0.09007368131876663	0.0	0.09007368131876663
spiders	0.06621496955185409	0.0	0.06621496955185409
palms	0.06617286150395034	0.0	0.06617286150395034

edge	0.058990744385672464	0.0	0.058990744385672464
ast	-0.056481838925186656	0.0	-0.056481838925186656
alt	-0.05257654040287	0.0	-0.05257654040287
platelet	0.05126314745980294	0.0	0.05126314745980294
...			

The default measure algorithm for multi-fault diagnosis uses the maximum magnitude of probability change over all faults and all observation outcomes. Our last diagnosis will keep the observations, but the measure algorithm will be normalized Euclidean distance in the fault probability vector space.

Python:

```
diag.set_multi_fault_algorithm(pysmile.MultiFaultAlgorithmType.L2_NORMALIZED_DISTANCE)
```

Java:

```
diag.setMultiFaultAlgorithm(DiagNetwork.MultiFaultAlgorithmType.L2_NORMALIZED_DISTANCE);
```

C#:

```
diag.MultiFaultAlgorithm = DiagNetwork.MultiFaultAlgorithmType.L2NormalizedDistance;
```

R:

```
diag$setMultiFaultAlgorithm(diag$MultiFaultAlgorithmType$L2_NORMALIZED_DISTANCE)
```

Here's the observation output after `DiagNetwork.update`:

Observations:			
Node Id	Measure	Cost	InfoGain
age	0.5131574568755791	0.0	0.5131574568755791
ast	0.36613951542960194	0.0	0.36613951542960194
alt	0.3263962662652687	0.0	0.3263962662652687
triglycerides	0.29763021106578275	0.0	0.29763021106578275
inr	0.23813502209247214	0.0	0.23813502209247214
alcoholism	0.23215767497663264	0.0	0.23215767497663264
bilirubin	0.2217351591088665	0.0	0.2217351591088665
hepatomegaly	0.21201689659082573	0.0	0.21201689659082573
ggtp	0.20797749804060328	0.0	0.20797749804060328
cholesterol	0.20654898750297776	0.0	0.20654898750297776
irregular_liver	0.20400352015821183	0.0	0.20400352015821183
ESR	0.19743075347810374	0.0	0.19743075347810374
spleen	0.18236843341781642	0.0	0.18236843341781642
albumin	0.18078914435171048	0.0	0.18078914435171048
...			

This concludes tutorial 9.

7.9.1 Tutorial9.py

```
import pysmile

# Tutorial9 loads HeparII.xdsl file
# and runs multiple diagnostic algorithms
# Use the link below to download the HeparII.xdsl file:
# https://support.bayesfusion.com/docs/Examples/Discrete%20Bayesian%20Networks/HeparII.xdsl

class Tutorial9:
    def __init__(self):
        print("Starting tutorial 9...")
        net = pysmile.Network()
        net.read_file("HeparII.xdsl")
        print("Hepar model loaded")
        print_diag_types(net)

        print("Creating diagnostic network object")
        diag = pysmile.DiagNetwork(net)
        print_fault_indices(diag)
```

```

pursued_fault_idx = diag.get_pursued_fault()
print(f"The default (most likely) pursued fault is at index {pursued_fault_idx}: "
      + f"{diag.get_fault_node_id(pursued_fault_idx)}"
      + f"={diag.get_fault_outcome_id(pursued_fault_idx)}")

print("Running diagnosis with no instantiated observations")
diag_results = diag.update()
print_diag_results(net, diag_results)

print("Running diagnosis with three observations")
diag.instantiate_observation("jaundice", "present")
diag.instantiate_observation("nausea", "absent")
diag.instantiate_observation("obesity", "present")
diag_results = diag.update()
print_diag_results(net, diag_results)

print("Running diagnosis with two observations and focusing on Hyperbilirubinemia")
diag.release_observation("nausea")
diag.instantiate_observation("obesity", "absent")
hyperbilirubinemia_idx = diag.get_fault_index("Hyperbilirubinemia", "present")
diag.set_pursued_fault(hyperbilirubinemia_idx)
diag_results = diag.update()
print_diag_results(net, diag_results)

print("Switching algorithm to cross-entropy, observations and pursued fault unchanged")
diag.set_single_fault_algorithm(pysmile.SingleFaultAlgorithmType.CROSSENTROPY)
diag_results = diag.update()
print_diag_results(net, diag_results)

print("Running diagnosis with two observations and "
      + "focusing on both Hyperbilirubinemia and Steatosis")
steatosis_idx = diag.get_fault_index("Steatosis", "present")
diag.set_pursued_faults([steatosis_idx, hyperbilirubinemia_idx])
diag_results = diag.update()
print_diag_results(net, diag_results)

print("Switching algorithm to L2 distance, observations and pursued faults unchanged")
diag.set_multi_fault_algorithm(pysmile.MultiFaultAlgorithmType.L2_NORMALIZED_DISTANCE)
diag_results = diag.update()
print_diag_results(net, diag_results)

print("Tutorial 9 complete.")

def print_diag_types(net):
    print(f"Network has {net.get_node_count()} nodes")
    print("{:<20} {:<30} {:<30}".format("Node Id", "Diagnostic Type", "Fault Outcomes"))
    for node_id in net.get_all_node_ids():
        diag_type = net.get_node_diag_type(node_id)
        fault_outcomes = []
        if diag_type == pysmile.NodeDiagType.FAULT:
            for outcome_id in net.get_outcome_ids(node_id):
                if net.is_fault_outcome(node_id, outcome_id):
                    fault_outcomes.append(outcome_id)
        print("{:<20} {:<30} {:<30}".format(node_id, str(net.get_node_diag_type(node_id)),
                                           str(fault_outcomes) if diag_type == pysmile.NodeDiagType.FAULT else "n/a"))

def print_fault_indices(diag):
    fault_count = diag.get_fault_count()
    print(f"Diagnostic network has {fault_count} faults (node/outcome pairs)")
    print("{:<12} {:<20} {:<20}".format("Fault Index", "Fault Node Id", "Fault Outcome Id"))

```

```

    for fidx in range(fault_count):
        print("{:<12} {:<20} {:<20}".format(
            fidx, diag.get_fault_node_id(fidx), diag.get_fault_outcome_id(fidx)))

def print_fault_info(net, fault):
    print("{:<20} {:<20} {:<24} {:<8}".format(
        net.get_node_id(fault.node), net.get_outcome_id(fault.node, fault.outcome),
        fault.probability, "Yes" if fault.is_pursued else "No"))

def print_observation_info(net, observation):
    print("{:<20} {:<24} {:<8} {:<20}".format(net.get_node_id(observation.node),
        observation.measure, observation.cost, observation.info_gain))

def print_diag_results(net, diag_results):
    print("Diag results start\n\nFaults:")
    print("{:<20} {:<20} {:<24} {:<8}".format(
        "Node Id", "Outcome Id", "Probability", "Is Pursued"))
    for fault_info in diag_results.faults:
        print_fault_info(net, fault_info)
    print("\nObservations:")
    print("{:<20} {:<24} {:<8} {:<20}".format("Node Id", "Measure", "Cost", "InfoGain"))
    for observation_info in diag_results.observations:
        print_observation_info(net, observation_info)
    print("\nDiag results end\n")

```

7.9.2 Tutorial9.java

```

package tutorials;

import smile.*;

import java.util.ArrayList;
import java.util.List;

// Tutorial9 loads HeparII.xdsl file
// and runs multiple diagnostic algorithms
// Use the link below to download the HeparII.xdsl file:
// https://support.bayesfusion.com/docs/Examples/Discrete%20Bayesian%20Networks/HeparII.xdsl

public class Tutorial9 {
    public static void run() {
        System.out.println("Starting diagnosis example");
        Network net = new Network();
        net.readFile("HeparII.xdsl");
        System.out.println("Hepar model loaded");
        printDiagTypes(net);

        System.out.println("Creating diagnostic network object");
        DiagNetwork diag = new DiagNetwork(net);
        printFaultIndices(diag);

        int pursuedFaultIdx = diag.getPursuedFault();
        System.out.printf("The default (most likely) pursued fault is at index %d: %s=%s\n",
            pursuedFaultIdx, diag.getFaultNodeId(pursuedFaultIdx),
            diag.getFaultOutcomeId(pursuedFaultIdx));

        System.out.println("Running diagnosis with no instantiated observations");
        DiagResults diagResults = diag.update();
        printDiagResults(net, diagResults);
    }
}

```

```

        System.out.println("Running diagnosis with three observations");
        diag.instantiateObservation("jaundice", "present");
        diag.instantiateObservation("nausea", "absent");
        diag.instantiateObservation("obesity", "present");
        diagResults = diag.update();
        printDiagResults(net, diagResults);

        System.out.println("Running diagnosis with two observations and "
            + "focusing on Hyperbilirubinemia");
        diag.releaseObservation("nausea");
        diag.instantiateObservation("obesity", "absent");
        int hyperbilirubinemiaIdx = diag.getFaultIndex("Hyperbilirubinemia", "present");
        diag.setPursuedFault(hyperbilirubinemiaIdx);
        diagResults = diag.update();
        printDiagResults(net, diagResults);

        System.out.println("Switching algorithm to cross-entropy, "
            + "observations and pursued fault unchanged");
        diag.setSingleFaultAlgorithm(DiagNetwork.SingleFaultAlgorithmType.CROSSENTROPY);
        diagResults = diag.update();
        printDiagResults(net, diagResults);

        System.out.println("Running diagnosis with two observations and "
            + "focusing on both Hyperbilirubinemia and Steatosis");
        int steatosisIdx = diag.getFaultIndex("Steatosis", "present");
        diag.setPursuedFaults(new int[]{steatosisIdx, hyperbilirubinemiaIdx});
        diagResults = diag.update();
        printDiagResults(net, diagResults);

        System.out.println("Switching algorithm to L2 distance, "
            + "observations and pursued faults unchanged");
        diag.setMultiFaultAlgorithm(DiagNetwork.MultiFaultAlgorithmType.L2_NORMALIZED_DISTANCE);
        diagResults = diag.update();
        printDiagResults(net, diagResults);

        System.out.println("Tutorial9 complete.");
    }

    public static void printDiagTypes(Network net) {
        System.out.printf("Network has %d nodes\n", net.getNodeCount());
        System.out.printf("%-20s %-30s %-30s\n",
            "Node Id", "Diagnostic Type", "Fault Outcomes");
        String[] nodeIds = net.getAllNodeIds();
        for (String nodeId : nodeIds) {
            int diagType = net.getNodeDiagType(nodeId);
            List<String> faultOutcomes = new ArrayList<>();
            if (diagType == Network.NodeDiagType.FAULT) {
                for (String outcomeId : net.getOutcomeIds(nodeId)) {
                    if (net.isFaultOutcome(nodeId, outcomeId)) {
                        faultOutcomes.add(outcomeId);
                    }
                }
            }
            System.out.printf("%-20s %-30s %-30s\n", nodeId, diagType,
                (faultOutcomes.size() != 0) ? String.join(", ", faultOutcomes) : "n/a");
        }
    }

    public static void printFaultIndices(DiagNetwork diag) {
        int faultCount = diag.getFaultCount();
        System.out.printf("Diagnostic network has %d faults (node/outcome pairs)\n",

```

```

        faultCount);
    System.out.printf("%-12s %-20s %-20s\n",
        "Fault Index", "Fault Node Id", "Fault Outcome Id");
    for (int fIdx = 0; fIdx < faultCount; fIdx++) {
        System.out.printf("%-12d %-20s %-20s\n",
            fIdx, diag.getFaultNodeId(fIdx), diag.getFaultOutcomeId(fIdx));
    }
}

public static void printFaultInfo(Network net, FaultInfo fault) {
    System.out.printf("%-20s %-20s %-24s %-8s\n",
        net.getNodeId(fault.node), net.getOutcomeId(fault.node, fault.outcome),
        fault.probability, fault.isPursued ? "Yes" : "No");
}

public static void printObservationInfo(Network net, ObservationInfo observation) {
    System.out.printf("%-20s %-24s %-8s %-20s\n",
        net.getNodeId(observation.node), observation.measure,
        observation.cost, observation.infoGain);
}

public static void printDiagResults(Network net, DiagResults diagResults) {
    System.out.println("Diag results start\n\nFaults:");
    System.out.printf("%-20s %-20s %-24s %-8s\n",
        "Node Id", "Outcome Id", "Probability", "Is Pursued");
    for (FaultInfo faultInfo : diagResults.faults) {
        printFaultInfo(net, faultInfo);
    }
    System.out.println("\nObservations:");
    System.out.printf("%-20s %-24s %-8s %-20s\n",
        "Node Id", "Measure", "Cost", "InfoGain");
    for (ObservationInfo observationInfo : diagResults.observations) {
        printObservationInfo(net, observationInfo);
    }
    System.out.println("\nDiag results end\n");
}
}

```

7.9.3 Tutorial9.cs

```

using System;
using System.Collections.Generic;
using Smile;

// Tutorial9 loads HeparII.xdsl file
// and runs multiple diagnostic algorithms
// Use the link below to download the HeparII.xdsl file:
// https://support.bayesfusion.com/docs/Examples/Discrete%20Bayesian%20Networks/HeparII.xdsl

namespace SmileNetTutorial
{
    class Tutorial9
    {
        public static void Run()
        {
            Console.WriteLine(System.IO.Directory.GetCurrentDirectory());
            Console.WriteLine("Starting Tutorial9...");
            Network net = new Network();
            net.ReadFile("HeparII.xdsl");
            Console.WriteLine("Hepar model loaded");
        }
    }
}

```

```

PrintDiagTypes(net);

Console.WriteLine("Creating diagnostic network object");
DiagNetwork diag = new DiagNetwork(net);
PrintFaultIndices(diag);

int pursuedFaultIdx = diag.GetPursuedFault();
Console.WriteLine("The default (most likely) pursued fault is at index {0}: {1}={2}",
    pursuedFaultIdx,
    diag.GetFaultNodeId(pursuedFaultIdx), diag.GetFaultOutcomeId(pursuedFaultIdx));

Console.WriteLine("Running diagnosis with no instantiated observations");
DiagResults diagResults = diag.Update();
PrintDiagResults(net, diagResults);

Console.WriteLine("Running diagnosis with three observations");
diag.InstantiateObservation("jaundice", "present");
diag.InstantiateObservation("nausea", "absent");
diag.InstantiateObservation("obesity", "present");
diagResults = diag.Update();
PrintDiagResults(net, diagResults);

Console.WriteLine("Running diagnosis with two observations and "
    + "focusing on Hyperbilirubinemia");
diag.ReleaseObservation("nausea");
diag.InstantiateObservation("obesity", "absent");
int hyperbilirubinemiaIdx = diag.GetFaultIndex("Hyperbilirubinemia", "present");
diag.SetPursuedFault(hyperbilirubinemiaIdx);
diagResults = diag.Update();
PrintDiagResults(net, diagResults);

Console.WriteLine("Switching algorithm to cross-entropy, "
    + "observations and pursued fault unchanged");
diag.SingleFaultAlgorithm = DiagNetwork.SingleFaultAlgorithmType.Crossentropy;
diagResults = diag.Update();
PrintDiagResults(net, diagResults);

Console.WriteLine("Running diagnosis with two observations and "
    + "focusing on both Hyperbilirubinemia and Steatosis");
int steatosisIdx = diag.GetFaultIndex("Steatosis", "present");
diag.SetPursuedFaults(new int[] { steatosisIdx, hyperbilirubinemiaIdx });
diagResults = diag.Update();
PrintDiagResults(net, diagResults);

Console.WriteLine("Switching algorithm to L2 distance, "
    + "observations and pursued faults unchanged");
diag.MultiFaultAlgorithm = DiagNetwork.MultiFaultAlgorithmType.L2NormalizedDistance;
diagResults = diag.Update();
PrintDiagResults(net, diagResults);

Console.WriteLine("Tutorial9 complete.");
}

public static void PrintDiagTypes(Network net)
{
    Console.WriteLine("Network has {0} nodes", net.GetNodeCount());
    Console.WriteLine("{0,-20} {1,-30} {2,-30}",
        "Node Id", "Diagnostic Type", "Fault Outcomes");
    string[] nodeIds = net.GetAllNodeIds();
    foreach (string nodeId in nodeIds)
    {

```

```

        Network.NodeDiagType diagType = net.GetNodeDiagType(nodeId);
        List<string> faultOutcomes = new List<string>();
        if (diagType == Network.NodeDiagType.Fault)
        {
            foreach (string outcomeId in net.GetOutcomeIds(nodeId))
            {
                if (net.IsFaultOutcome(nodeId, outcomeId))
                {
                    faultOutcomes.Add(outcomeId);
                }
            }
        }
        Console.WriteLine("{0,-20} {1,-30} {2,-30}", nodeId, diagType,
            (faultOutcomes.Count != 0) ? string.Join(" ", faultOutcomes) : "n/a");
    }
}

public static void PrintFaultIndices(DiagNetwork diag)
{
    int faultCount = diag.FaultCount;
    Console.WriteLine("Diagnostic network has {0} faults (node/outcome pairs)",
        faultCount);
    Console.WriteLine("{0,-12} {1,-20} {2,-20}",
        "Fault Index", "Fault Node Id", "Fault Outcome Id");
    for (int fIdx = 0; fIdx < faultCount; fIdx++)
    {
        Console.WriteLine("{0,-12} {1,-20} {2,-20}", fIdx,
            diag.GetFaultNodeId(fIdx), diag.GetFaultOutcomeId(fIdx));
    }
}

public static void PrintFaultInfo(Network net, FaultInfo fault)
{
    Console.WriteLine("{0,-20} {1,-20} {2,-24} {3,-8}",
        net.GetNodeId(fault.Node), net.GetOutcomeId(fault.Node, fault.Outcome),
        fault.Probability, fault.IsPursued ? "Yes" : "No");
}

public static void PrintObservationInfo(Network net, ObservationInfo observation)
{
    Console.WriteLine("{0,-20} {1,-24} {2,-8} {3,-20}",
        net.GetNodeId(observation.Node), observation.Measure,
        observation.Cost, observation.InfoGain);
}

public static void PrintDiagResults(Network net, DiagResults diagResults)
{
    Console.WriteLine("Diag results start\n\nFaults:");
    Console.WriteLine("{0,-20} {1,-20} {2,-24} {3,-8}",
        "Node Id", "Outcome Id", "Probability", "Is Pursued");
    foreach (FaultInfo faultInfo in diagResults.Faults)
    {
        PrintFaultInfo(net, faultInfo);
    }
    Console.WriteLine("\nObservations:");
    Console.WriteLine("{0,-20} {1,-24} {2,-8} {3,-20}",
        "Node Id", "Measure", "Cost", "InfoGain");
    foreach (ObservationInfo observationInfo in diagResults.Observations)
    {
        PrintObservationInfo(net, observationInfo);
    }
}

```

```

        Console.WriteLine("\nDiag results end\n");
    }
}

```

7.9.4 Tutorial9.R

```

library(rSMILE)
source("License.R")

# Tutorial9 loads HeparII.xdsl file
# and runs multiple diagnostic algorithms
# Use the link below to download the HeparII.xdsl file:
# https://support.bayesfusion.com/docs/Examples/Discrete%20Bayesian%20Networks/HeparII.xdsl

printDiagTypes <- function(net) {
  cat(sprintf("Network has %d nodes\n", net$getNodeCount()))
  cat(sprintf("%-20s %-30s %-30s\n", "Node Id", "Diagnostic Type", "Fault Outcomes"))
  nodeIds <- net$getAllNodeIds()
  for (nodeId in nodeIds) {
    diagType <- net$getNodeDiagType(nodeId)
    faultOutcomes <- character()
    if (diagType == net$NodeDiagType$FAULT) {
      outcomeIds <- net$getOutcomeIds(nodeId)
      for (outcomeId in outcomeIds) {
        if (net$isFaultOutcome(nodeId, outcomeId)) {
          faultOutcomes <- c(faultOutcomes, outcomeId)
        }
      }
    }
    cat(sprintf("%-20s %-30s %-30s\n",
      nodeId, as.character(net$getNodeDiagType(nodeId)),
      ifelse(diagType == net$NodeDiagType$FAULT,
        paste(faultOutcomes, collapse = ", "), "n/a")))
  }
}

printFaultIndices <- function(diag) {
  faultCount <- diag$getFaultCount()
  cat(sprintf("Diagnostic network has %d faults (node/outcome pairs)\n", faultCount))
  cat(sprintf("%-12s %-20s %-20s\n", "Fault Index", "Fault Node Id", "Fault Outcome Id"))
  for (fIdx in 0:(faultCount-1)) {
    cat(sprintf("%-12d %-20s %-20s\n", fIdx, diag
      $getNodeId(fIdx), diag$getOutcomeId(fIdx)))
  }
}

printFaultInfo <- function(net, fault) {
  cat(sprintf("%-20s %-20s %-24s %-8s\n",
    net$getNodeId(fault$node), net$getOutcomeId(fault$node, fault$outcome),
    fault$probability, ifelse(fault$isPursued, "Yes", "No")))
}

printObservationInfo <- function(net, observation) {
  cat(sprintf("%-20s %-24s %-8s %-20s\n", net$getNodeId(observation$node),
    observation$measure, observation$cost, observation$infoGain))
}

printDiagResults <- function(net, diagResults) {
  cat("\n-- Diag results start\n\nFaults:\n")

```

```

cat(sprintf("%-20s %-20s %-24s %-8s\n",
  "Node Id", "Outcome Id", "Probability", "Is Pursued"))
for (faultInfo in diagResults$faults) {
  printFaultInfo(net, faultInfo)
}
cat("\nObservations:\n")
cat(sprintf("%-20s %-24s %-8s %-20s\n",
  "Node Id", "Measure", "Cost", "InfoGain"))
for (observationInfo in diagResults$observations) {
  printObservationInfo(net, observationInfo)
}
cat("\nDiag results end\n")
}

cat("Starting Tutorial9\n")
net <- Network()
net$readFile("HeparII.xdsl")
cat("Hepar model loaded\n")
printDiagTypes(net)

cat("Creating diagnostic network object\n")
diag <- DiagNetwork(net)
printFaultIndices(diag)

pursuedFaultIdx <- diag$getPursuedFault()
cat(sprintf("The default (most likely) pursued fault is at index %d: %s=%s\n",
  pursuedFaultIdx,
  diag$getNodeId(pursuedFaultIdx), diag$getNodeOutcomeId(pursuedFaultIdx)))

cat("Running diagnosis with no instantiated observations\n")
diagResults <- diag$update()
printDiagResults(net, diagResults)

cat("Running diagnosis with three observations\n")
diag$instantiateObservation("jaundice", "present")
diag$instantiateObservation("nausea", "absent")
diag$instantiateObservation("obesity", "present")
diagResults <- diag$update()
printDiagResults(net, diagResults)

cat("Running diagnosis with two observations and focusing on Hyperbilirubinemia\n")
diag$releaseObservation("nausea")
diag$instantiateObservation("obesity", "absent")
hyperbilirubinemiaIdx <- diag$getNodeIndex("Hyperbilirubinemia", "present")
diag$setPursuedFault(hyperbilirubinemiaIdx)
diagResults <- diag$update()
printDiagResults(net, diagResults)

cat("Switching algorithm to cross-entropy, observations and pursued fault unchanged")
diag$setSingleFaultAlgorithm(diag$SingleFaultAlgorithmType$CROSSENTROPY)
diagResults <- diag$update()
printDiagResults(net, diagResults)

cat("Running diagnosis with two observations",
  "and focusing on both Hyperbilirubinemia and Steatosis\n")
steatosisIdx <- diag$getNodeIndex("Steatosis", "present")
diag$setPursuedFaults(c(steatosisIdx, hyperbilirubinemiaIdx))
diagResults <- diag$update()
printDiagResults(net, diagResults)

cat("Switching algorithm to L2 distance, observations and pursued faults unchanged")

```

```
diag$setMultiFaultAlgorithm(diag$MultiFaultAlgorithmType$L2_NORMALIZED_DISTANCE)
diagResults <- diag$update()
printDiagResults(net, diagResults)

cat("Tutorial9 complete.\n")
```

7.10 Tutorial 10: Structure learning

Suppose we have a file *Credit10K.csv* consisting of 10,000 records of customers collected at a bank. Each of these customers was measured on several variables, *Payment History*, *Work History*, *Reliability*, *Debit*, *Income*, *Ratio of Debts to Income*, *Assets*, *Worth*, *Profession*, *Future Income*, *Age* and *Credit Worthiness*. The file is too large to include in this manual, but it can be downloaded from

<https://support.bayesfusion.com/docs/Examples/Learning/Credit10K.csv>. It is also included in the *Examples\Learning* directory of GeNIe installation. The first few records of the file (included among the example files with GeNIe distribution) look as follows in GeNIe:

	PaymentHistory	WorkHistory	Reliability	Debit	Income	RatioDebInc	Assets	Worth	Profession	FutureIncome	Age	CreditWorthiness
►	Without_ Reference	Unstable	Unreliable	a0_11100	s70001_more	Favorable	wealthy	High	Medium_income_profession	Promissing	a16_21	Negative
	Acceptable	Unjustified_no_work	Unreliable	a0_11100	s70001_more	Favorable	average	High	Medium_income_profession	Promissing	a66_up	Negative
	Acceptable	Unstable	Reliable	a25901_more	s30001_70000	Unfavorable	wealthy	High	Low_income_profession	Not_promissing	a16_21	Negative
	Excellent	Unstable	Reliable	a25901_more	s30001_70000	Unfavorable	average	Medium	Medium_income_profession	Not_promissing	a16_21	Negative
	Excellent	Unjustified_no_work	Unreliable	a11101_25900	s0_30000	Unfavorable	average	Low	Medium_income_profession	Not_promissing	a66_up	Negative
	Without_ Reference	Stable	Reliable	a0_11100	s30001_70000	Favorable	average	High	Medium_income_profession	Promissing	a16_21	Positive
	NoAcceptable	Stable	Unreliable	a0_11100	s70001_more	Favorable	wealthy	High	Medium_income_profession	Promissing	a66_up	Positive
	Excellent	Stable	Reliable	a0_11100	s70001_more	Favorable	wealthy	High	Low_income_profession	Promissing	a66_up	Positive
	Excellent	Stable	Reliable	a25901_more	s70001_more	Unfavorable	poor	High	Low_income_profession	Not_promissing	a16_21	Negative
	NoAcceptable	Stable	Unreliable	a0_11100	s30001_70000	Favorable	average	Medium	Medium_income_profession	Promissing	a22_65	Positive
	Without_ Reference	Justified_no_work	Reliable	a25901_more	s70001_more	Unfavorable	poor	High	Low_income_profession	Not_promissing	a16_21	Negative
	NoAcceptable	Unstable	Unreliable	a25901_more	s30001_70000	Unfavorable	wealthy	High	Medium_income_profession	Promissing	a16_21	Negative
	NoAcceptable	Justified_no_work	Unreliable	a25901_more	s30001_70000	Unfavorable	wealthy	High	High_income_profession	Promissing	a22_65	Negative
	Excellent	Stable	Reliable	a11101_25900	s0_30000	Unfavorable	average	Low	Medium_income_profession	Not_promissing	a16_21	Negative
	Acceptable	Stable	Unreliable	a25901_more	s0_30000	Unfavorable	wealthy	Medium	Medium_income_profession	Not_promissing	a66_up	Negative
	Without_ Reference	Unjustified_no_work	Unreliable	a0_11100	s0_30000	Favorable	poor	Low	Low_income_profession	Not_promissing	a66_up	Positive
	Acceptable	Unstable	Reliable	a11101_25900	s30001_70000	Unfavorable	average	Medium	Low_income_profession	Not_promissing	a66_up	Negative
	Without_ Reference	Unstable	Unreliable	a0_11100	s30001_70000	Unfavorable	average	High	Medium_income_profession	Promissing	a16_21	Negative

We want to learn the structure of the Bayesian network based on the data. Our program attempts to load the data file from disk, and prints the number of data set variables and records if loading was successful. For clarity, we omit the exception handling in the code snippets here. The tutorial programs include the exception handlers for each learning call.

Python:

```
ds = pysmile.learning.DataSet()
ds.read_file("Credit10k.csv")
print(f"Dataset has {ds.get_variable_count()} variables (columns)
      and {ds.get_record_count()} records (rows)")
```

Java:

```
DataSet ds = new DataSet();
ds.readFile("Credit10k.csv");
System.out.println(String.format(
    "Dataset has %d variables (columns) and %d records (rows)",
    ds.getVariableCount(), ds.getRecordCount()));
```

C#:

```
DataSet ds = new DataSet();
ds.ReadFile("Credit10k.csv");
Console.WriteLine(String.Format(
    "Dataset has {0} variables (columns) and {1} records (rows)",
    ds.VariableCount, ds.RecordCount));
```

R:

```
ds <- DataSet()
ds$readFile("Credit10k.csv")
cat(sprintf("Dataset has %d variables (columns) and %d records (rows)\n",
```

```
ds$getVariableCount(), ds$getRecordCount()))
```

The most general structure learning algorithm is Bayesian Search. It is a hill climbing procedure with random restarts, guided by log-likelihood scoring heuristic. Its search space is hyper-exponential, so for best results the algorithm can be fine-tuned with multiple with multiple options (details are available in the SMILE manual's *Reference* section). We set the number of iterations (random restarts) to 50, and fix the random number seed to ensure reproducible results. The `learn` method requires an input data set. The learning algorithm will perform 50 iterations, each starting with randomized structure, and compute a log-likelihood score for at the end of each iteration. The best structure will be copied into the output network, and parameter learning will be run internally. The output network will therefore be ready to use.

Python:

```
bayes_search = pysmile.learning.BayesianSearch()
bayes_search.set_iteration_count(50)
bayes_search.set_rand_seed(9876543)
net1 = bayes_search.learn(ds)
```

Java:

```
BayesianSearch bayesSearch = new BayesianSearch();
bayesSearch.setIterationCount(50);
bayesSearch.setRandSeed(9876543);
Network net1 = bayesSearch.learn(ds);
```

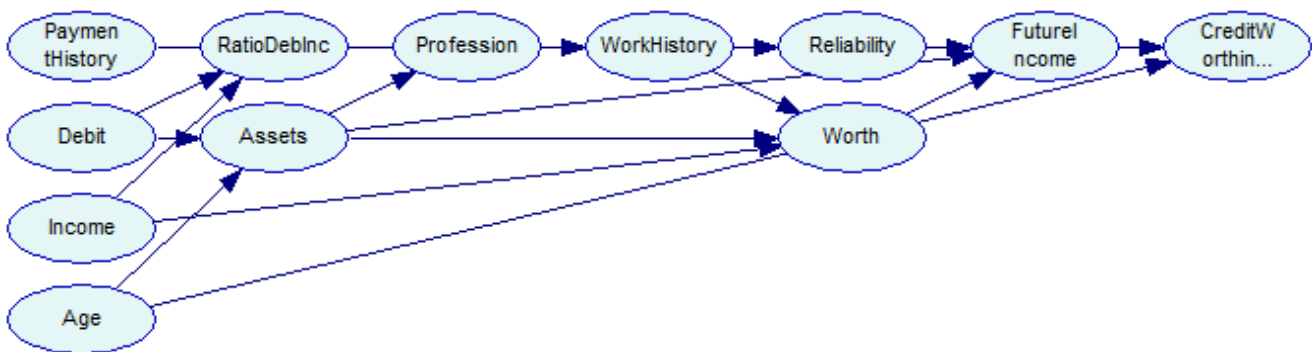
C#:

```
BayesianSearch bayesSearch = new BayesianSearch();
bayesSearch.IterationCount = 50;
bayesSearch.RandSeed = 9876543;
Network net1 = bayesSearch.Learn(ds);
```

R:

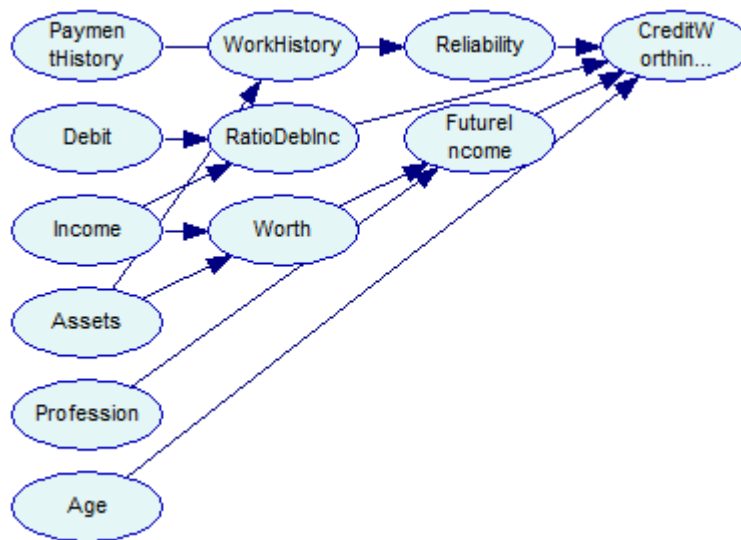
```
bayesSearch <- BayesianSearch()
bayesSearch$setIterationCount(50)
bayesSearch$setRandSeed(9876543)
net1 <- bayesSearch$learn(ds)
```

The output network looks like this in GeNIe:



Please note that the nodes in the network learned in SMILE occupy the same screen position. The image below shows the network after the parent ordering layout has been applied. We plan to include simple layout as the last step of the learning in the upcoming release of SMILE.

The network is saved for reference in the *tutorial9-bs1.xdsl* file. We can obtain a different structure by changing the seed for the random number generator and calling `learn` again. The selected seed value is 3456789, and the output looks like this:



The tutorial saves this network to *tutorial9-bs2.xdsl* proceeds to third attempt at Bayesian Search, this time with background knowledge. We want to forbid the arc between *Age* and *CreditWorthiness*, and force an arc between *Age* and *Profession*. When specifying the background knowledge, we refer to the data set variables as their indices, so we need to convert the textual identifiers to indices first.

Python:

```

idx_age = ds.find_variable("Age")
idx_profession = ds.find_variable("Profession")
idx_credit_worthiness = ds.find_variable("CreditWorthiness")
background_knowledge = pysmile.learning.BkKnowledge()
background_knowledge.match_data(ds)
background_knowledge.add_forbidden_arc(idx_age, idx_credit_worthiness)
background_knowledge.add_forced_arc(idx_age, idx_profession)
bayes_search.set_bk_knowledge(background_knowledge)
net3 = bayes_search.learn(ds)

```

Java:

```

int idxAge = ds.findVariable("Age");
int idxProfession = ds.findVariable("Profession");
int idxCreditWorthiness = ds.findVariable("CreditWorthiness");
BkKnowledge backgroundKnowledge = new BkKnowledge();
backgroundKnowledge.matchData(ds);
backgroundKnowledge.addForbiddenArc(idxAge, idxCreditWorthiness);
backgroundKnowledge.addForcedArc(idxAge, idxProfession);
bayesSearch.setBkKnowledge(backgroundKnowledge);
Network net3 = bayesSearch.Learn(ds);

```

C#:

```

int idxAge = ds.FindVariable("Age");
int idxProfession = ds.FindVariable("Profession");
int idxCreditWorthiness = ds.FindVariable("CreditWorthiness");
BkKnowledge backgroundKnowledge = new BkKnowledge();
backgroundKnowledge.MatchData(ds);
backgroundKnowledge.AddForbiddenArc(idxAge, idxCreditWorthiness);
backgroundKnowledge.AddForcedArc(idxAge, idxProfession);
bayesSearch.BkKnowledge = backgroundKnowledge;
Network net3 = bayesSearch.Learn(ds);

```

R:

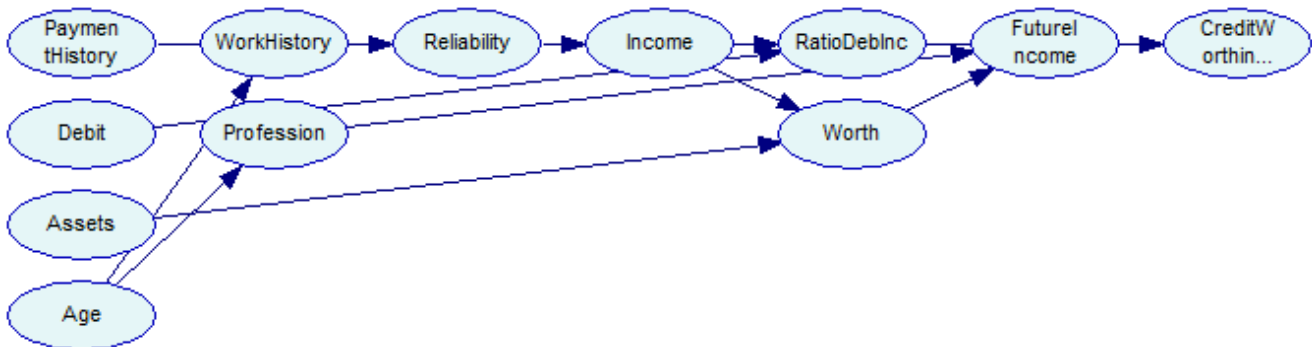
```

idxAge <- ds$findVariable("Age")
idxProfession <- ds$findVariable("Profession")
idxCreditWorthiness <- ds$findVariable("CreditWorthiness")

```

```
backgroundKnowledge <- BkKnowledge()
backgroundKnowledge$matchData(ds)
backgroundKnowledge$addForbiddenArc(idxAge, idxCreditWorthiness)
backgroundKnowledge$addForcedArc(idxAge, idxProfession)
bayesSearch$setBkKnowledge(backgroundKnowledge)
net3 <- bayesSearch$learn(ds)
```

The output network now looks as follows - please note the absence of an arc from *Age* to *CreditWorthiness* and the forced arc between *Age* and *Profession*. This network is saved to *tutorial9-bs3.xdsl*.



We now change our learning algorithm to Tree Augmented Naive Bayes (TAN). The TAN algorithm starts with a Naive Bayes structure (i.e., one in which the class variable is the only parent of all remaining, feature variables) and adds connections between the feature variables to account for possible dependence between them, conditional on the class variable. The algorithm imposes the limit of only one additional parent of every feature variable (additional to the class variable, which is a parent of every feature variable). We need to specify the class variable with its textual identifier. In this example *CreditWorthiness* is the class variable.

Python:

```
tan = pysmile.learning.TAN()
tan.set_rand_seed(777999)
tan.set_class_variable_id("CreditWorthiness")
net4 = tan.learn(ds)
```

Java:

```
Network net4;
TAN tan = new TAN();
tan.setRandSeed(777999);
tan.setClassVariableId("CreditWorthiness");
net4 = tan.learn(ds);
```

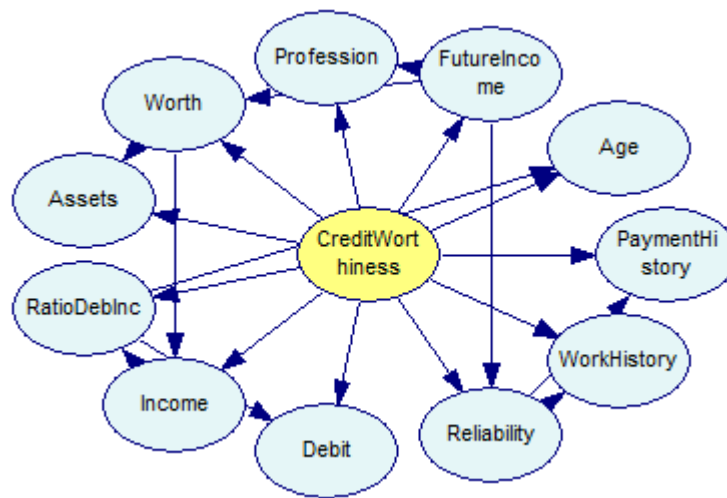
C#:

```
Network net4;
TAN tan = new TAN();
tan.RandSeed = 777999;
tan.ClassVariableId = "CreditWorthiness";
net4 = tan.Learn(ds);
```

R:

```
tan <- TAN()
tan$setRandSeed(777999)
tan$setClassVariableId("CreditWorthiness")
net4 <- tan$learn(ds)
```

The program saves the network for reference in *tutorial9-tan.xdsl* file. The network looks as follows (the class node has yellow background):



Our final structure learning algorithm will be PC, which uses independences observed in data (established by means of classical independence tests) to infer the structure that has generated them. The output of the PC algorithm is a pattern, which is an adjacency matrix, and does not necessarily represent a directed acyclic graph (DAG). The pattern can be converted to network with the `makeNetwork` method, which enforces the DAG criterion on the pattern, and copies the variables and edges to the network.

Python:

```
pc = pysmile.learning.PC()
pattern = pc.learn(ds)
net5 = pattern.make_network(ds)
```

Java:

```
PC pc = new PC();
Pattern pattern = pc.learn(ds);
Network net5 = pattern.makeNetwork(ds);
```

C#:

```
PC pc = new PC();
Pattern pattern = pc.Learn(ds);
Network net5 = pattern.MakeNetwork(ds);
```

R:

```
pc <- PC()
pattern <- pc$learn(ds)
net5 <- pattern$makeNetwork(ds)
```

At this point our output network has nodes with outcomes based on the labels in the data set, but the probability distributions in node definitions are uniform over outcomes. To complete the learning process, we need to manually call a parameter learning procedure implemented in the EM class.

Python:

```
em = pysmile.learning.EM()
matching = ds.match_network(net5)
em.set_uniformize_parameters(False)
em.set_randomize_parameters(False)
em.set_eq_sample_size(0)
em.learn(ds, net5, matching)
```

Java:

```
EM em = new EM();
DataMatch[] matching = ds.matchNetwork(net5);
em.setUniformizeParameters(false);
em.setRandomizeParameters(false);
em.setEqSampleSize(0);
```

```
em.learn(ds, net5, matching);
```

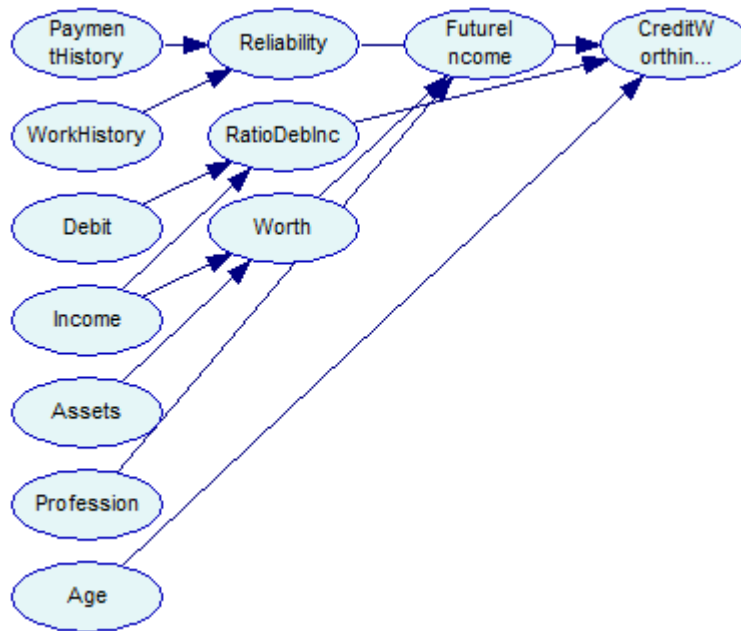
C#:

```
EM em = new EM();
DataMatch[] matching = ds.MatchNetwork(net5);
em.UniformizeParameters = false;
em.RandomizeParameters = false;
em.EqSampleSize = 0;
em.Learn(ds, net5, matching);
```

R:

```
em <- EM()
matching <- ds$matchNetwork(net5)
em$setUniformizeParameters(FALSE)
em$setRandomizeParameters(FALSE)
em$setEqSampleSize(0)
em$learn(ds, net5, matching)
```

Note the matchNetwork call, which populates the matching variable using identifiers of data set variables and network nodes. This information is subsequently used by EM. Learn to associate data set variables with network node. The output network looks like as follows:



The network is saved for reference in the *tutorial9-pc-em.xdsl* file and Tutorial 9 finishes.

7.10.1 Tutorial10.py

```
import pysmile

# Tutorial10 loads Credit10k.csv file
# and runs multiple structure learning algorithms
# using the loaded dataset.
# Use the link below to download the Credit10k.csv file:
# https://support.bayesfusion.com/docs/Examples/Learning/Credit10K.csv

class Tutorial10:
    def __init__(self):
        print("Starting tutorial10...")
        ds = pysmile.learning.DataSet()
```

```

try:
    ds.read_file("Credit10k.csv")
except pysmile.SMILEException:
    print("Dataset load failed")
    return
print(f"Dataset has {ds.get_variable_count()} variables (columns) "
      + f"and {ds.get_record_count()} records (rows)")
bayes_search = pysmile.learning.BayesianSearch()
bayes_search.set_iteration_count(50)
bayes_search.set_rand_seed(9876543)
try:
    net1 = bayes_search.learn(ds)
except pysmile.SMILEException:
    print("Bayesian Search failed")
    return
print(f"1st Bayesian Search finished, structure score: {bayes_search.get_last_score()}")
net1.write_file("tutorial10-bs1.xdsl")

bayes_search.set_rand_seed(3456789)
try:
    net2 = bayes_search.learn(ds)
except pysmile.SMILEException:
    print("Bayesian Search failed")
    return
print(f"2nd Bayesian Search finished, structure score: {bayes_search.get_last_score()}")
net2.write_file("tutorial10-bs2.xdsl")

idx_age = ds.find_variable("Age")
idx_profession = ds.find_variable("Profession")
idx_credit_worthiness = ds.find_variable("CreditWorthiness")

if idx_age < 0 or idx_profession < 0 or idx_credit_worthiness < 0:
    print("Can't find dataset variables for background knowledge")
    print("The loaded file may not be Credit10k.csv")
    return
background_knowledge = pysmile.learning.BkKnowledge()
background_knowledge.match_data(ds)
background_knowledge.add_forbidden_arc(idx_age, idx_credit_worthiness)
background_knowledge.add_forced_arc(idx_age, idx_profession)

bayes_search.set_bk_knowledge(background_knowledge)
try:
    net3 = bayes_search.learn(ds)
except pysmile.SMILEException:
    print("Bayesian Search failed")
    return
print(f"3rd Bayesian Search finished, structure score: {bayes_search.get_last_score()}")
net3.write_file("tutorial10-bs3.xdsl")

tan = pysmile.learning.TAN()
tan.set_rand_seed(777999)
tan.set_class_variable_id("CreditWorthiness")
try:
    net4 = tan.learn(ds)
except pysmile.SMILEException:
    print("TAN failed")
    return
print("Tree-augmented Naive Bayes finished")
net4.write_file("tutorial10-tan.xdsl")

pc = pysmile.learning.PC()

```

```

try:
    pattern = pc.learn(ds)
except pysmile.SMILEException:
    print("PC failed")
    return
net5 = pattern.make_network(ds)
print("PC finished, proceeding to parameter learning")
net5.write_file("tutorial10-pc.xdsl")
em = pysmile.learning.EM()
try:
    matching = ds.match_network(net5)
except pysmile.SMILEException:
    print("Can't automatically match network with dataset")
    return
em.set_uniformize_parameters(False)
em.set_randomize_parameters(False)
em.set_eq_sample_size(0)
try:
    em.learn(ds, net5, matching)
except pysmile.SMILEException:
    print("EM failed")
    return
print("EM finished")
net5.write_file("tutorial10-pc-em.xdsl")

print("Tutorial10 complete.")

```

7.10.2 Tutorial10.java

```

package tutorials;

import smile.*;
import smile.learning.*;

// Tutorial9 loads Credit10k.csv file
// and runs multiple structure learning algorithms
// using the loaded dataset.
// Use the link below to download the Credit10k.csv file:
// https://support.bayesfusion.com/docs/Examples/Learning/Credit10K.csv

public class Tutorial10 {
    public static void run() {
        System.out.println("Starting Tutorial9...");
        DataSet ds = new DataSet();
        try
        {
            ds.readFile("Credit10k.csv");
        } catch (SMILEException ex)
        {
            System.out.println("Dataset load failed");
            return;
        }
        System.out.printf("Dataset has %d variables (columns) and %d records (rows)\n",
            ds.getVariableCount(), ds.getRecordCount());
        BayesianSearch bayesSearch = new BayesianSearch();
        bayesSearch.setIterationCount(50);
        bayesSearch.setRandSeed(9876543);
        Network net1;
        try
        {

```

```
        net1 = bayesSearch.learn(ds);
    } catch (SMILEException ex)
    {
        System.out.println("Bayesian Search failed");
        return;
    }
    System.out.printf("1st Bayesian Search finished, structure score: %f\n",
        bayesSearch.getLastScore());
    net1.writeFile("tutorial9-bs1.xdsl");

    Network net2;
    bayesSearch.setRandSeed(3456789);
    try
    {
        net2 = bayesSearch.learn(ds);
    }
    catch (SMILEException ex)
    {
        System.out.println("Bayesian Search failed");
        return;
    }
    System.out.printf("2nd Bayesian Search finished, structure score: %f\n",
        bayesSearch.getLastScore());
    net2.writeFile("tutorial9-bs2.xdsl");

    int idxAge = ds.findVariable("Age");
    int idxProfession = ds.findVariable("Profession");
    int idxCreditWorthiness = ds.findVariable("CreditWorthiness");
    if (idxAge < 0 || idxProfession < 0 || idxCreditWorthiness < 0)
    {
        System.out.println("Can't find dataset variables for background knowledge");
        System.out.println("The loaded file may not be Credit10k.csv");
        return;
    }

    BkKnowledge backgroundKnowledge = new BkKnowledge();
    backgroundKnowledge.matchData(ds);
    backgroundKnowledge.addForbiddenArc(idxAge, idxCreditWorthiness);
    backgroundKnowledge.addForcedArc(idxAge, idxProfession);
    bayesSearch.setBkKnowledge(backgroundKnowledge);
    Network net3;
    try
    {
        net3 = bayesSearch.learn(ds);
    }
    catch (SMILEException ex)
    {
        System.out.println("Bayesian Search failed");
        return;
    }
    System.out.printf("3rd Bayesian Search finished, structure score: %f\n",
        bayesSearch.getLastScore());
    net3.writeFile("tutorial9-bs3.xdsl");

    Network net4;
    TAN tan = new TAN();
    tan.setRandSeed(777999);
    tan.setClassVariableId("CreditWorthiness");
    try
    {
        net4 = tan.learn(ds);
```

```

    }
    catch (SMILEException ex)
    {
        System.out.println("TAN failed");
        return;
    }
    System.out.println("Tree-augmented Naive Bayes finished");
    net4.writeFile("tutorial9-tan.xdsl");

    PC pc = new PC();
    Pattern pattern;
    try
    {
        pattern = pc.learn(ds);
    } catch (SMILEException ex)
    {
        System.out.println("PC failed");
        return;
    }
    Network net5 = pattern.makeNetwork(ds);
    System.out.println("PC finished, proceeding to parameter learning");
    net5.writeFile("tutorial9-pc.xdsl");
    EM em = new EM();
    DataMatch[] matching;
    try
    {
        matching = ds.matchNetwork(net5);
    }
    catch (SMILEException ex)
    {
        System.out.println("Can't automatically match network with dataset");
        return;
    }
    em.setUniformizeParameters(false);
    em.setRandomizeParameters(false);
    em.setEqSampleSize(0);
    try
    {
        em.learn(ds, net5, matching);
    }
    catch (SMILEException ex)
    {
        System.out.println("EM failed");
        return;
    }
    System.out.println("EM finished");
    net5.writeFile("tutorial9-pc-em.xdsl");

    System.out.println("Tutorial10 complete");
}
}

```

7.10.3 Tutorial10.cs

```

using System;
using Smile;
using Smile.Learning;

// Tutorial10 loads Credit10k.csv file
// and runs multiple structure learning algorithms

```

```
// using the loaded dataset.
// Use the link below to download the Credit10k.csv file:
// https://support.bayesfusion.com/docs/Examples/Learning/Credit10K.csv

namespace SmileNetTutorial
{
    class Tutorial10
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial10...");
            DataSet ds = new DataSet();
            try
            {
                ds.ReadFile("Credit10k.csv");
            } catch (SmileException)
            {
                Console.WriteLine("Dataset load failed");
                return;
            }
            Console.WriteLine("Dataset has {0} variables (columns) and {1} records (rows)",
                ds.VariableCount, ds.RecordCount);
            BayesianSearch bayesSearch = new BayesianSearch();
            bayesSearch.IterationCount = 50;
            bayesSearch.RandSeed = 9876543;
            Network net1;
            try
            {
                net1 = bayesSearch.Learn(ds);
            } catch (SmileException)
            {
                Console.WriteLine("Bayesian Search failed");
                return;
            }
            Console.WriteLine("1st Bayesian Search finished, structure score: {0}",
                bayesSearch.LastScore);
            net1.WriteFile("tutorial10-bs1.xdsl");

            Network net2;
            bayesSearch.RandSeed = 3456789;
            try
            {
                net2 = bayesSearch.Learn(ds);
            }
            catch (SmileException)
            {
                Console.WriteLine("Bayesian Search failed");
                return;
            }
            Console.WriteLine("2nd Bayesian Search finished, structure score: {0}",
                bayesSearch.LastScore);
            net2.WriteFile("tutorial10-bs2.xdsl");

            int idxAge = ds.FindVariable("Age");
            int idxProfession = ds.FindVariable("Profession");
            int idxCreditWorthiness = ds.FindVariable("CreditWorthiness");
            if (idxAge < 0 || idxProfession < 0 || idxCreditWorthiness < 0)
            {
                Console.WriteLine("Can't find dataset variables for background knowledge");
                Console.WriteLine("The loaded file may not be Credit10k.csv");
                return;
            }
        }
    }
}
```

```
}

BkKnowledge backgroundKnowledge = new BkKnowledge();
    backgroundKnowledge.MatchData(ds);
backgroundKnowledge.AddForbiddenArc(idxAge, idxCreditWorthiness);
backgroundKnowledge.AddForcedArc(idxAge, idxProfession);
bayesSearch.BkKnowledge = backgroundKnowledge;
Network net3;
try
{
    net3 = bayesSearch.Learn(ds);
}
catch (SmileException)
{
    Console.WriteLine("Bayesian Search failed");
    return;
}
Console.WriteLine("3rd Bayesian Search finished, structure score: {0}",
    bayesSearch.LastScore);
net3.WriteFile("tutorial10-bs3.xdsl");

Network net4;
TAN tan = new TAN();
tan.RandSeed = 777999;
tan.ClassVariableId = "CreditWorthiness";
try
{
    net4 = tan.Learn(ds);
}
catch (SmileException)
{
    Console.WriteLine("TAN failed");
    return;
}
Console.WriteLine("Tree-augmented Naive Bayes finished");
net4.WriteFile("tutorial10-tan.xdsl");

PC pc = new PC();
Pattern pattern;
try
{
    pattern = pc.Learn(ds);
} catch (SmileException)
{
    Console.WriteLine("PC failed");
    return;
}
Network net5 = pattern.MakeNetwork(ds);
Console.WriteLine("PC finished, proceeding to parameter learning");
net5.WriteFile("tutorial10-pc.xdsl");
EM em = new EM();
DataMatch[] matching;
try
{
    matching = ds.MatchNetwork(net5);
}
catch (SmileException)
{
    Console.WriteLine("Can't automatically match network with dataset");
    return;
}
```

```

        em.UniformizeParameters = false;
        em.RandomizeParameters = false;
        em.EqSampleSize = 0;
        try
        {
            em.Learn(ds, net5, matching);
        }
        catch (SmileException)
        {
            Console.WriteLine("EM failed");
            return;
        }
        Console.WriteLine("EM finished");
        net5.WriteFile("tutorial10-pc-em.xdsl");

        Console.WriteLine("Tutorial10 complete");
    }
}
}

```

7.10.4 Tutorial10.R

```

library(rSMILE)
source("License.R")

# Tutorial10 loads Credit10k.csv file
# and runs multiple structure learning algorithms
# using the loaded dataset.
# Use the link below to download the Credit10k.csv file:
# https://support.bayesfusion.com/docs/Examples/Learning/Credit10K.csv

cat("Starting Tutorial10...\n")
ds <- DataSet()

ds$readFile("Credit10k.csv")
cat(sprintf("Dataset has %d variables (columns) and %d records (rows)\n",
    ds$getVariableCount(), ds$getRecordCount()))
bayesSearch <- BayesianSearch()
bayesSearch$setIterationCount(50)
bayesSearch$setRandSeed(9876543)
net1 <- bayesSearch$learn(ds)
cat(sprintf("1st Bayesian Search finished, structure score: %f\n",
    bayesSearch$getLastScore()))
net1$writeFile("tutorial10-bs1.xdsl")

bayesSearch$setRandSeed(3456789)
net2 <- bayesSearch$learn(ds)
cat(sprintf("2nd Bayesian Search finished, structure score: %f\n",
    bayesSearch$getLastScore()))
net2$writeFile("tutorial10-bs2.xdsl")

idxAge <- ds$findVariable("Age")
idxProfession <- ds$findVariable("Profession")
idxCreditWorthiness <- ds$findVariable("CreditWorthiness")

backgroundKnowledge <- BkKnowledge()
backgroundKnowledge$matchData(ds)
backgroundKnowledge$addForbiddenArc(idxAge, idxCreditWorthiness)
backgroundKnowledge$addForcedArc(idxAge, idxProfession)

```

```
bayesSearch$setBkKnowledge(backgroundKnowledge)
net3 <- bayesSearch$learn(ds)
cat(sprintf("3rd Bayesian Search finished, structure score: %f\n",
  bayesSearch$getLastScore()))
net2$writeFile("tutorial10-bs3.xdsl")

tan <- TAN()
tan$setRandSeed(777999)
tan$setClassVariableId("CreditWorthiness")
net4 <- tan$learn(ds)
cat("Tree-augmented Naive Bayes finished")
net4$writeFile("tutorial10-tan.xdsl")

pc <- PC()
pattern <- pc$learn(ds)
net5 <- pattern$makeNetwork(ds)
cat("PC finished, proceeding to parameter learning\n")
net5$writeFile("tutorial10-pc.xdsl")
em <- EM()
matching <- ds$matchNetwork(net5)
em$setUniformizeParameters(FALSE)
em$setRandomizeParameters(FALSE)
em$setEqSampleSize(0)
em$learn(ds, net5, matching)
cat("EM finished\n")
net5$writeFile("tutorial10-pc-em.xdsl")

cat("Tutorial10 complete.\n")
```

Equations reference

8 Equations reference

The subsections of this chapter describe the notation used when specifying the definition for equation nodes and expression-based MAU nodes.

8.1 Operators

Arithmetic operators

- +** addition, e.g., if $x=3$ and $y=2$, $x+y$ produces 5
- subtraction and unary minus, e.g., if $x=3$ and $y=2$, $x-y$ produces 1 and $-x$ produces -3
- ^** exponentiation (a^b means a^b), e.g., if $x=3$ and $y=2$, x^y produces 9
- *** multiplication, e.g., if $x=3$ and $y=2$, $x*y$ produces 6
- /** division, e.g., if $x=3$ and $y=2$, x/y produces 1.5

Comparison operators

- >** greater than, e.g., if $x=3$ and $y=2$, $x>y$ produces 1
- <** smaller than, e.g., if $x=3$ and $y=2$, $x<y$ produces 0
- >=** greater or equal than, e.g., if $x=3$ and $y=2$, $x>=y$ produces 1
- <=** smaller or equal than, e.g., if $x=3$ and $y=2$, $x<=y$ produces 0
- <>** not equal, e.g., if $x=3$ and $y=2$, $x<>y$ produces 1
- =** equal, e.g., if $x=3$ and $y=2$, $x=y$ produces 0

Conditional selection operator

- ? :** ternary conditional operator like in C, C++ or Java programming languages.

This operator is essentially a shortcut to the *If* function. For example, $a=b?5:3$ is equivalent to $If(a=b, 5, 3)$.

Order of calculation, operator precedence, and parentheses

Expressions are evaluated from left to right, according to the precedence order specified below (1 denotes the highest precedence).

Precedence order	Operator
1	- (unary minus)
2	^ (exponentiation)
3	* and / (multiplicative operators)
4	+ and - (additive operators)
5	>, <, >=, <=, = (comparison operators)
6	?: (conditional selection)

To change the order of calculation, enclose in parentheses those parts of the formula that should be calculated first. This can be done recursively, i.e., parentheses can be nested indefinitely. For example, if $x=3$ and $y=2$, $2*y+x/3-y+1$ produces 4, $2*(y+x)/(3-y)+1$ produces 11, and $2*((y+x)/(3-y)+1)$ produces 12.

8.2 Random Number Generators

Random number generators **each generate a single sample** from the distributions defined below. In most equations, they can be imagined as random noise that distorts the equation. Because the fundamental algorithm for inference in continuous and hybrid models is stochastic simulation, it is possible to visualize what probability distributions these single samples result in for each of the variables in the model.

Caution: Probability distributions are not allowed in expression-based MAU nodes, as these are deterministic functions by definition.

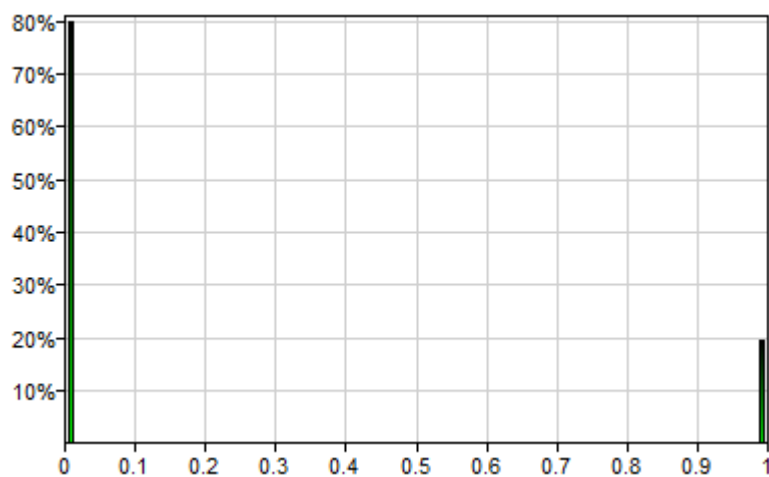
Choosing the right probability distribution over continuous data is a skill that requires some statistical insight. When the distribution is transformed by an equation expression, the task is daunting even for an experienced decision analyst. GeNIe offers an interactive tool for visualizing expressions, the Distribution Visualizer, with probability distributions through Monte Carlo simulation. The same functionality is available online at <https://prob.bayesfusion.com>. The screenshot below shows the online visualizer with the probability distribution sampled from the expression $\text{Log10}(\text{Sqrt}(\text{If}(\text{Uniform}(0,1)<1/4, \text{Lognormal}(-1,0.3), \text{Lognormal}(0,0.2))))$.



Bernoulli(p)

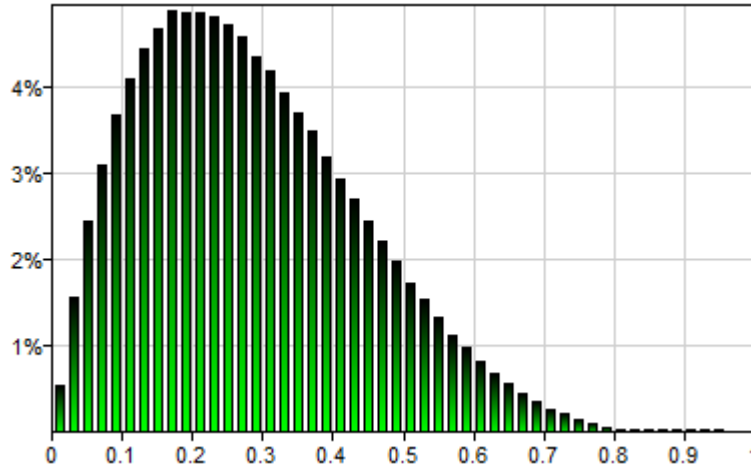
Bernoulli is a discrete distribution that generates 0 with probability $1-p$ and 1 with probability p .

$Bernoulli(0.2)$ will generate a single sample (0 or 1) from the following distribution, i.e., 1 with probability 0.2 and 0 with probability 0.8:



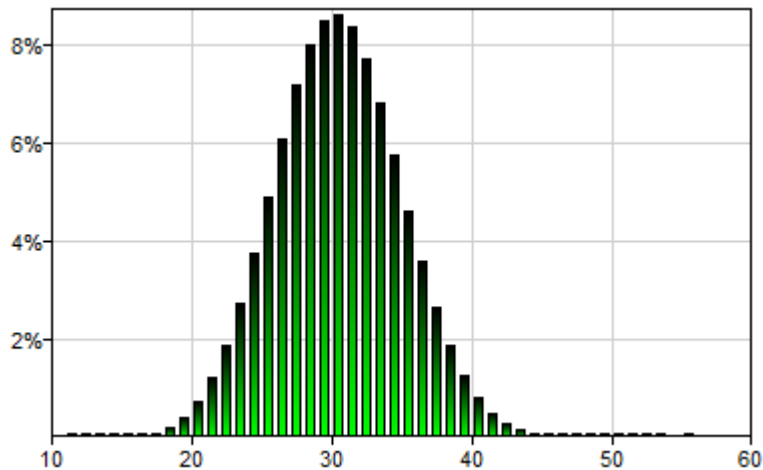
Beta(a,b)

The Beta distribution is a family of continuous probability distributions defined on the interval $[0, 1]$ and parametrized by two positive shape parameters, a and b (typically denoted by α and β), that control the shape of the distribution. $Beta(2, 5)$ will generate a single sample from the following distribution:



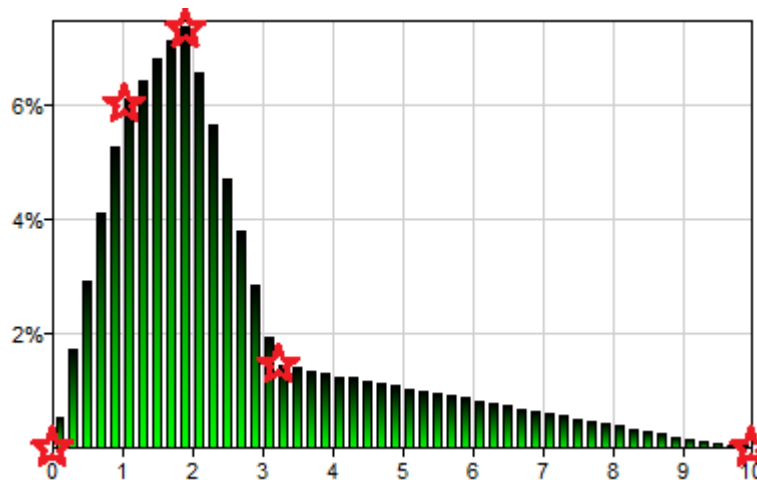
Binomial(n,p)

Binomial is a discrete probability distribution over the number of successes in a sequence of n independent trials, each of which yields a success with probability p . It will generate a single sample, which will be an integer number between 0 and n . A success/failure experiment is also called a Bernoulli trial. Hence, $Binomial(1, p)$ is equivalent to $Bernoulli(p)$. $Binomial(100, 0.3)$ will generate a single sample from the following distribution:



CustomPDF(x1,x2,...y1,y2,...)

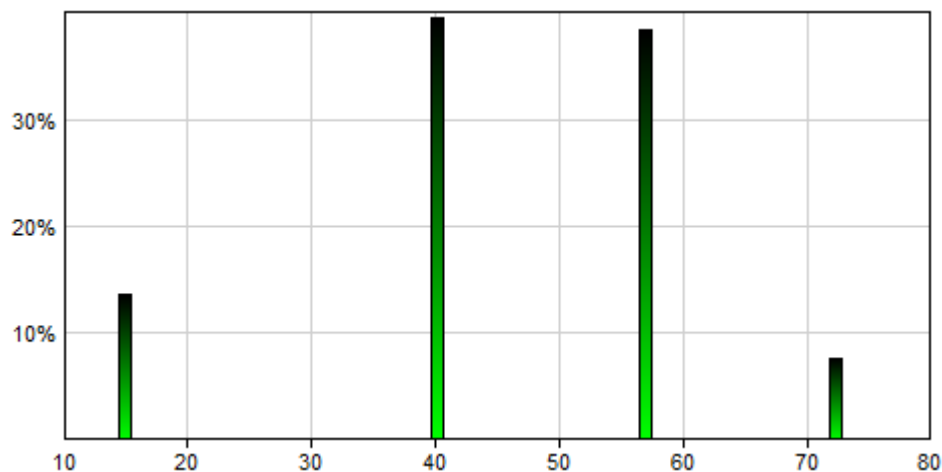
The CustomPDF distribution allows for specifying a non-parametric continuous probability distribution by means of a series of points on its probability density (PDF) function. Pairs (x_i, y_i) are coordinates of such points. The total number of parameters of CustomPDF function should thus to be even. Please note that x coordinates should be listed in increasing order. The PDF function specified does not need to be normalized, i.e., the area under the curve does not need to add up to 1.0. For example, $CustomPDF(0, 1.02, 1.9, 3.2, 10, 0, 4, 5, 1, 0)$ generates a single sample from the following distribution:



Stars on the plot mark the points defined by the *CustomPDF* arguments, i.e., (0, 0), (1.02, 4), (1.9, 5), (3.2, 1), and (10, 0).

Discrete(x1,x2,...,xn, p1,p2,...,pn)

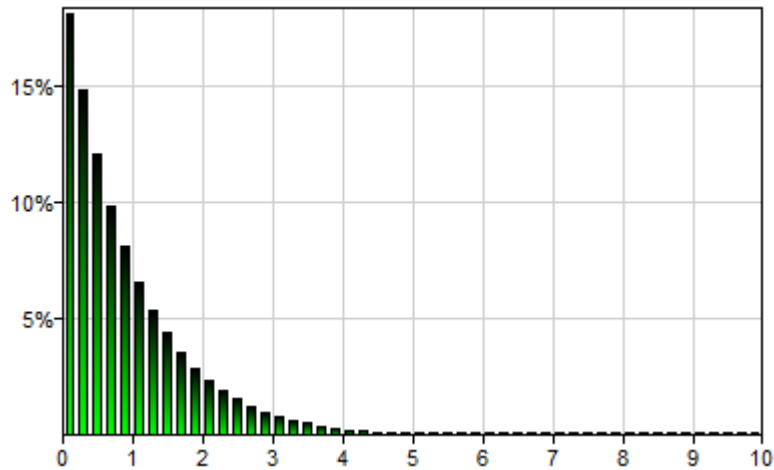
The Discrete distribution allows for specifying a discrete probability distribution over a collection of numerical values. It is one of the simplest random number generators, essentially replicating a discrete distribution and producing values $x_1, x_2, \dots, x_n$ with probabilities $p_1, p_2, \dots, p_n$. This distribution is useful in simulating a discrete node using an equation node. The total number of parameters of *Discrete()* function should be even. Please note that x values should be listed in increasing order. Even though the p values should in theory add up to 1.0 and we advise that they do, GeNIe perform normalization, i.e., modifies them proportionally to add up to 1.0. For example, *Discrete(15, 40, 57.5, 72.5, 0.137339, 0.397711, 0.387697, 0.0772532)* replicates the definition of the variable Age in the HeparII model and generates a single sample from the following distribution:



Exponential(lambda)

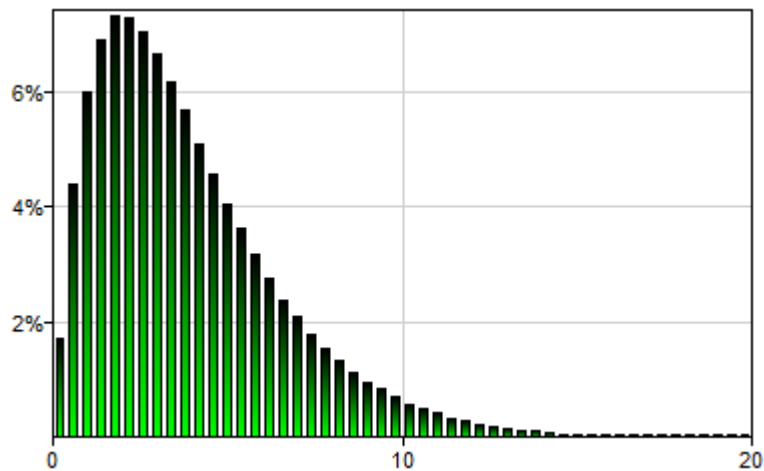
The exponential distribution is a continuous probability distribution that describes the time between events in a Poisson process, i.e., a process in which events occur continuously and independently at a constant average rate. Its only real-valued, positive parameter *lambda* (typically denoted by λ) determines the shape of the

distribution. It is a special case of the Gamma distribution. *Exponential(Lambda)* generates a single sample from the domain $(0, \infty)$. *Exponential(1)* will generate a single sample from the following distribution:



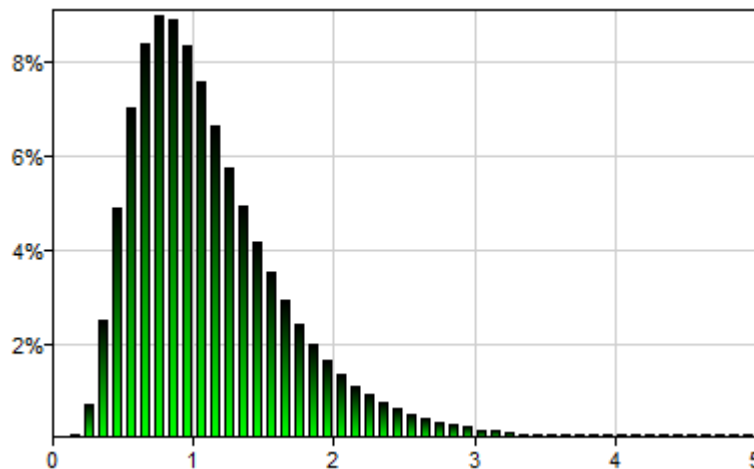
Gamma(shape,scale)

The Gamma distribution is a two-parameter family of continuous probability distributions. There are different parametrizations of the Gamma distribution in common use. SMILE parametrization follows one of the most popular parametrizations, with *shape* (often denoted by k) and *scale* (often denoted by θ) parameters, both positive real numbers. *Gamma(2.0, 2.0)* will generate a single sample from the following distribution:



Lognormal(mu,sigma)

The lognormal distribution is a continuous probability distribution of a random variable, whose logarithm is normally distributed. Thus, if a random variable X is lognormally distributed, then a variable $Y = \ln(X)$ has a normal distribution. Conversely, if Y has a normal distribution, then $X = e^Y$ has a lognormal distribution. A random variable which is lognormally distributed takes only positive values. *Lognormal(0, 0.5)* will generate a single sample from the following distribution:

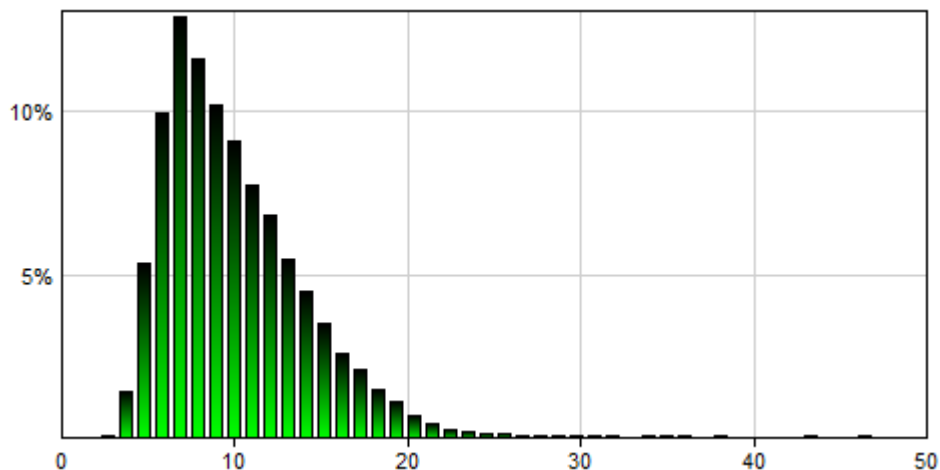


Metalog(lower,upper,k,x1,x2,...,y1,y2,...)

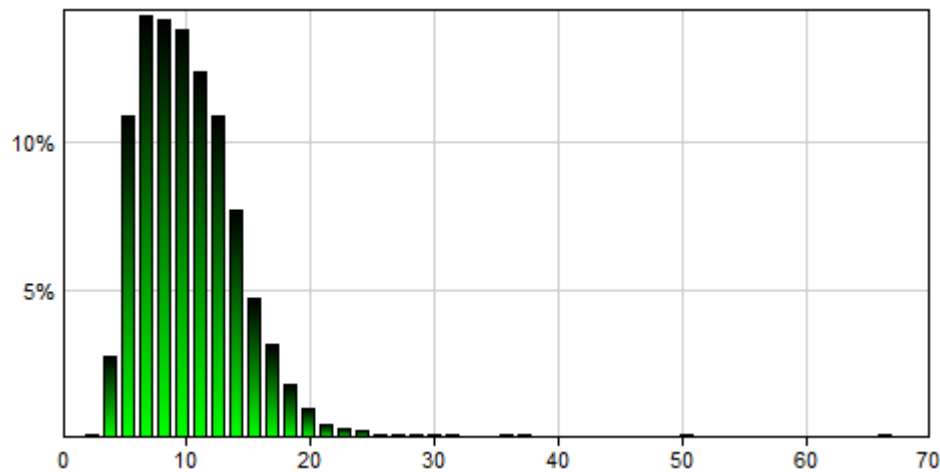
Metalog (also known as the Keelin) distribution is a very flexible distribution, capable of fitting many naturally occurring distributions. It can be specified by probability quantiles, which are values of the variable x_i and their corresponding cumulative probabilities y_i . Metalogs are able to represent distributions that are unbounded, semi-bounded, and bounded. lower and upper are the bounds of the distribution ($-Inf()$ and $Inf()$ denote lower and upper infinite bound respectively). k is a parameter of the metalog distribution, running from 2 to n , where n is the number of probability quantiles specified. Generally the higher the value of k , the more flexible the distribution but it is worth looking at the distributions generated for different values of k to find a compromise between complexity and goodness of fit. The choice of k is best performed interactively, looking at the family of metalog distributions generated from the probability quantiles. GeNIe contains a built-in tool, the Metalog Builder, which displays metalog PDF and CDF charts. This functionality is also available online at <https://metalog.bayesfusion.com>.

The examples below use identical bounds and probability quantiles with k equal 4, 6 and 8:

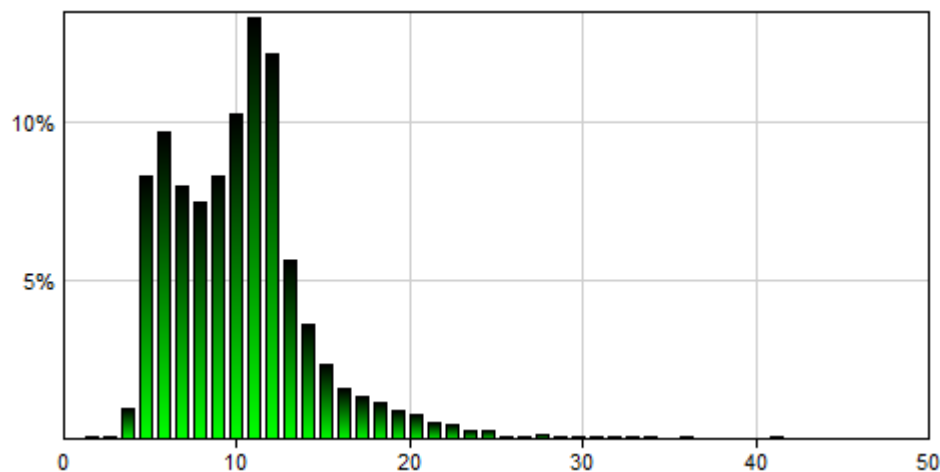
Metalog(0, Inf(), 4, 3, 4, 5, 5, 7, 10, 12, 15, 18, 32, 0.001, 0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99) will generate a single sample from the following distribution:



Metalog(0, Inf(), 6, 3, 4, 5, 5, 7, 10, 12, 15, 18, 32, 0.001, 0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99) will generate a single sample from the following distribution:



MetaLog(θ , *Inf*(), 8, 3, 4, 5, 5, 7, 10, 12, 15, 18, 32, 0.001, 0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99) will generate a single sample from the following distribution:

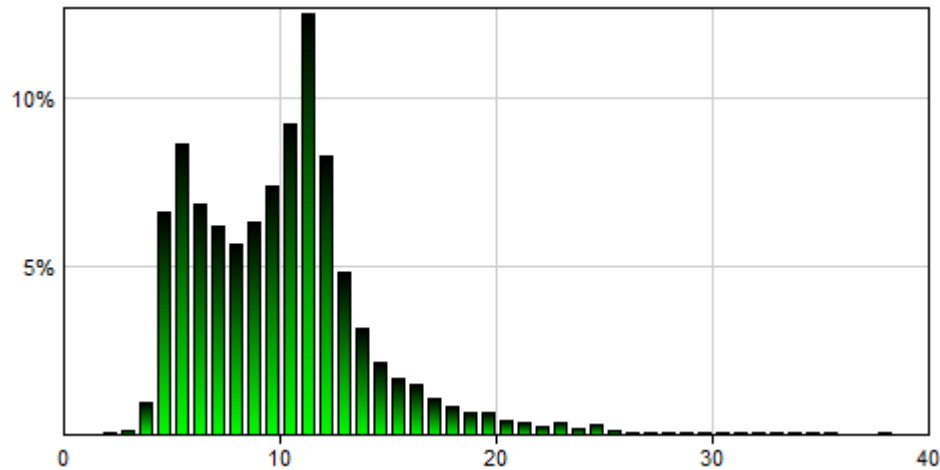


For more information about metalog distributions, please look at the comprehensive article on the topic on Wikipedia (https://en.wikipedia.org/wiki/Metalog_distribution), the Metalog Distribution web site created by Tom Keelin (<http://metalogdistributions.com/>) or the Metalog Distributions YouTube channel (<https://www.youtube.com/channel/UCyHZ5neKhV1mSsedzDBoqyA>).

MetalogA(lower,upper,a1,a2,...)

MetaLogA function uses what one could call internal metalog coefficients (a_i and, additionally, the *Lower* and *upper* bound of the distribution) that, contrary to percentiles of the distribution used as parameters of *MetaLog*, do not have easily interpretable meaning. One might expect that *MetaLogA* is more efficient in sample generation, as it skips the whole process of deriving the distribution from which it subsequently generates a sample. However, SMILE has an efficient caching scheme that makes *MetaLog* equally efficient in practice.

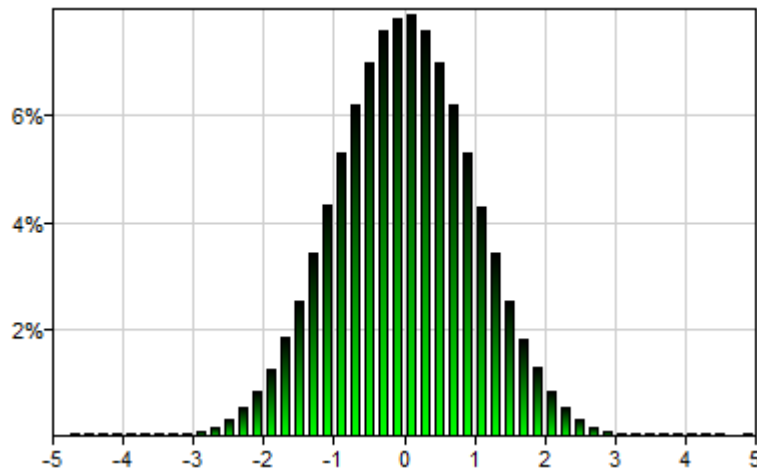
MetaLogA(θ , *Inf*(), 2.30769, 0.164148, -0.731388, 0.343231, 0.883249, -0.170727, 1.40341, 2.64853), which is equivalent to *MetaLog*(θ , *Inf*(), 8, 3, 4, 5, 5, 7, 10, 12, 15, 18, 32, 0.001, 0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99), will generate a single sample from the following distribution:



Please note that the number of a_i parameters of *MetaLogA* is the same as the k parameter in *Metalog*. Obtaining the parameters a_i outside of tools like Metalog Builder is rather challenging.

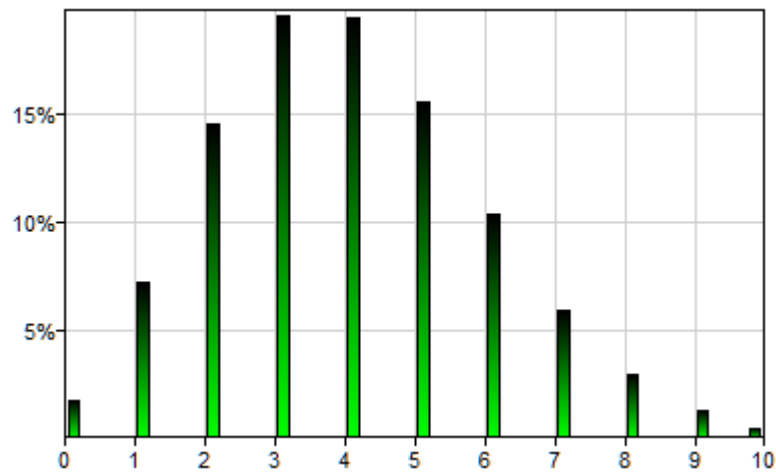
Normal(μ, σ)

Normal (also known as Gaussian) distribution is the most commonly occurring continuous probability distribution. It is symmetric and defined over the real domain. Its two parameters, μ (mean, μ) and σ (standard deviation, σ), control the position of its mode and its spread respectively. *Normal*($\theta, 1$) will generate a single sample from the following distribution:



Poisson(λ)

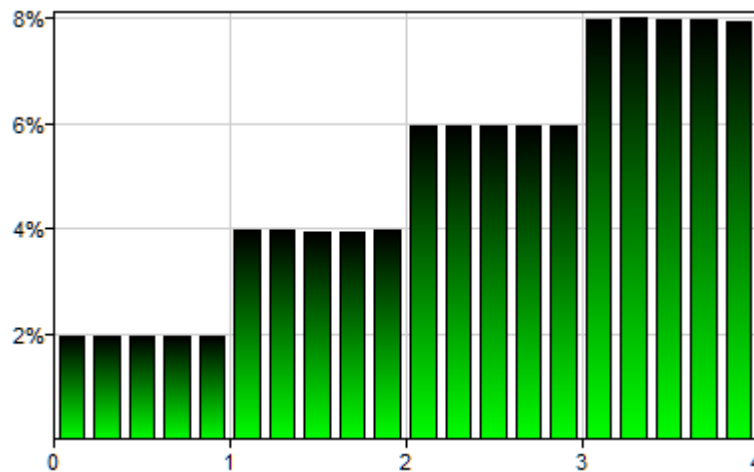
Poisson distribution is a discrete probability distribution typically used to express the probability of a given number of events occurring in a fixed interval of time or space if these events occur with a known constant rate and independently of the time since the last event. Its only parameter, λ , is the expected number of occurrences (which does not need to be integer). *Poisson*(4) will generate a single sample from the following distribution:



Steps(x1,x2,...y1,y2,...)

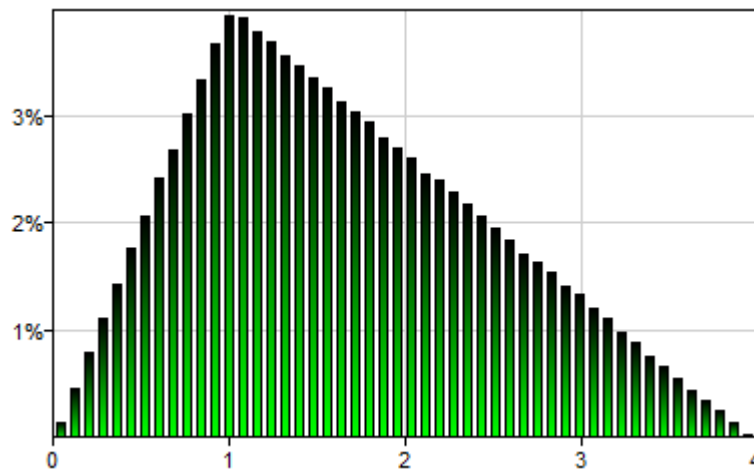
The *Steps* distribution allows for specifying a non-parametric continuous probability distribution by means of a series of steps on its probability density (PDF) function. It is similar to the *CustomPDF* function, although it does not specify the inflection points but rather intervals and the height of a step-wise probability distribution in each of the intervals. Because the number of interval borders is always one more than the number of intervals between them, the total number of parameters of *Steps* function should be odd. Please note that x coordinates should be listed in increasing order. The PDF function specified does not need to be normalized, i.e., the area under the curve does not need to add up to 1.0.

Example: *Steps(0,1,2,3,4,1,2,3,4)* generates a single sample from the following distribution::



Triangular(min,mod,max)

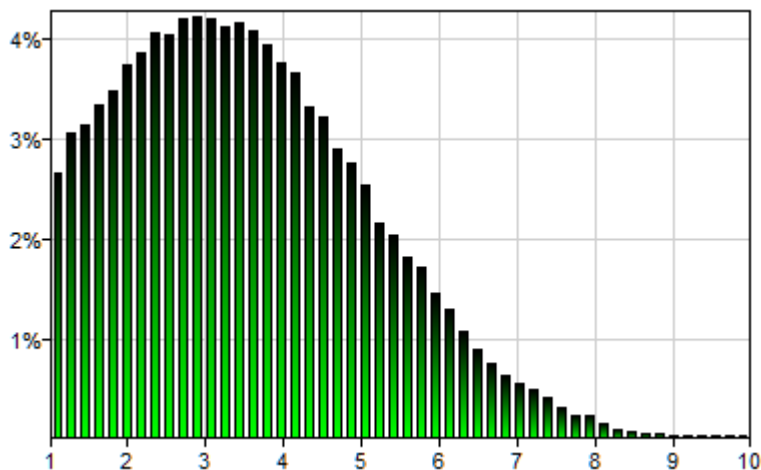
Triangular distribution is a continuous probability distribution with lower limit *min*, upper limit *max* and mode *mod*, where $min \leq mod \leq max$. *Triangular(0,1,3)* will generate a single sample from the following distribution:



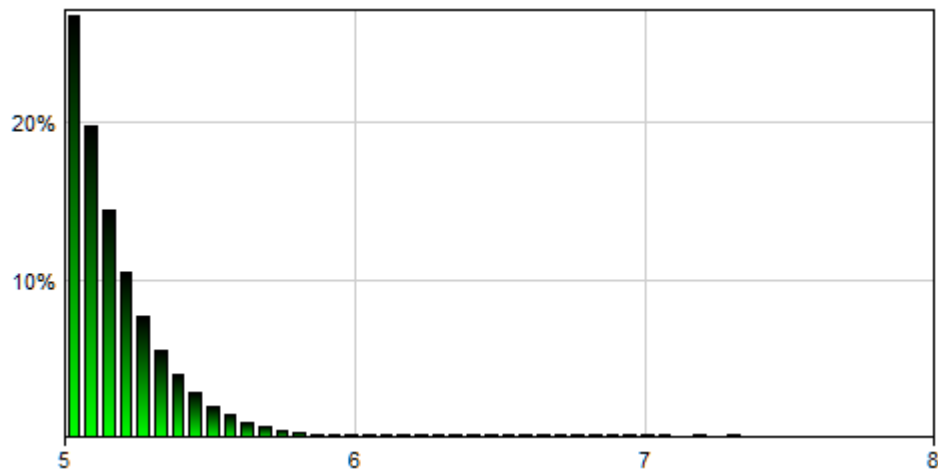
`TruncNormal(mu,sigma,lower[,upper])`

Truncated Normal distribution is essentially a Normal distribution that is truncated at the values lower and upper. This distribution is especially useful in situation when we want to limit physically impossible values in the model.

TruncNormal(3,2,1) will generate a single sample from the following distribution:

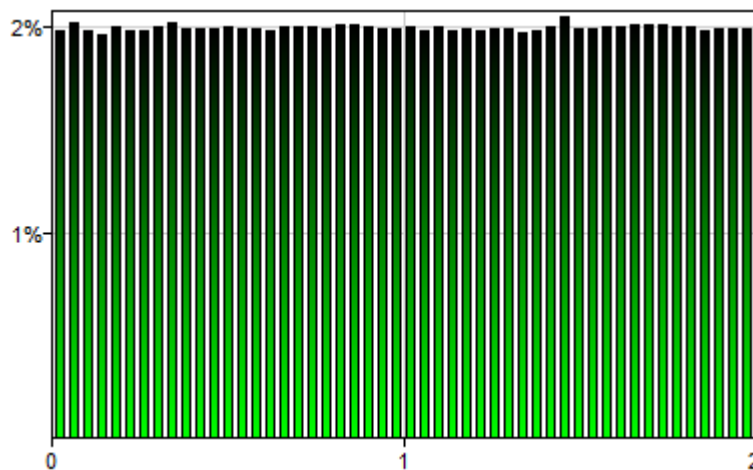


TruncNormal(0,1,5,10) will generate a single sample from the following distribution:



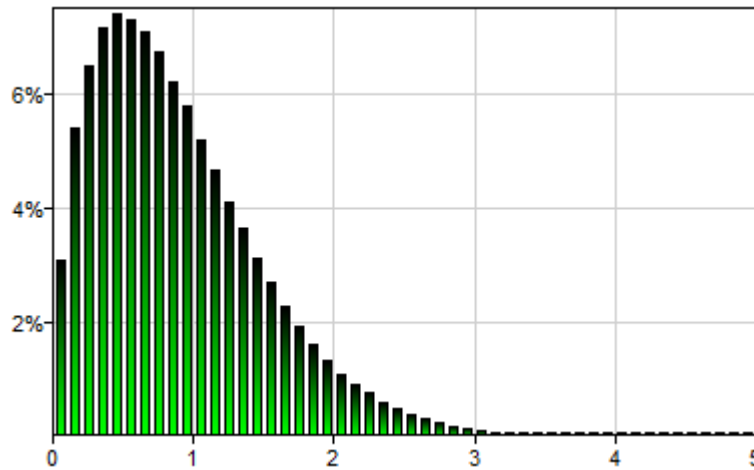
Uniform(a,b)

The continuous uniform distribution, also known as the rectangular distribution, is a family of probability distributions under which any two intervals of the same length are equally probable. It is defined two parameters, a and b , which are the minimum and the maximum values of the random variable. $Uniform(0, 2)$ will generate a single sample from the following distribution:



Weibull(lambda,k)

Weibull distribution is a continuous probability distribution named after a Swedish mathematician Waloddi Weibull, used in modeling such phenomena as particle size. It is characterized by two positive real parameters: the scale parameter λ and the shape parameter k . $Weibull(1, 1.5)$ will generate a single sample from the following distribution:



8.3 Statistical Functions

Erf(x)

Returns error function ((also called the Gauss error function, inverse of the Normal PDF), e.g., $\text{Erf}(1.0) = 0.842701$).

NormalPDF(x, mu, sigma)

Returns the value of PDF for the normal distribution given by mu and sigma at x.

NormalCDF(x, mu, sigma)

Returns the value of CDF for the normal distribution given by mu and sigma at x.

8.4 Arithmetic Functions

Abs(x)

Returns the absolute value of a number, e.g., $\text{Abs}(5.3) = \text{Abs}(-5.3) = 5.3$.

Exp(x)

Returns e (Euler's number) raised to the power of x , e.g., $\text{Exp}(2.2) = e^{2.2} = 9.02501$.

Gammaln(x)

Returns the natural logarithm of the Gamma function ($\Gamma(x)$), e.g., $\text{Gammaln}(2.2) = 0.0969475$.

GCD(n,k)

Returns the greatest common integer divisor of its two integer arguments n and k , e.g., $\text{GCD}(15, 25) = 5$. When the arguments are not integers, their fractional part is ignored.

Inf()

Returns positive infinity. For negative infinity, please use `-Inf()`.

LCM(n,k)

Returns the least common integer multiple of its two integer arguments n and k , e.g., $\text{LCM}(15, 25)=75$. When the arguments are not integers, their fractional part is ignored.

Ln(x)

Returns the natural logarithm of x , which has to be non-negative, e.g., $\text{Ln}(10)=2.30259$.

Log(x,b)

Returns the base b logarithm of x , which has to be non-negative, e.g., $\text{Log}(10, 2)=3.32193$.

Log10(x)

Returns decimal logarithm of x , which has to be non-negative, e.g., $\text{Log10}(100)=2$.

Mod(x,y)

Returns the remainder of dividing x by y . This works for both integer and real arguments, e.g., $\text{Mod}(5, 2)=1$, $\text{Mod}(9.2, 3.7)=1.8$.

Pow10(x)

Returns 10 raised to the power of x , e.g., $\text{Pow10}(2)=10^2=100$.

Round(x)

Returns the integer that is nearest to x , e.g., $\text{Round}(2.2)=2$, $\text{Round}(3.5)=4$.

Sign(x)

Returns 1 if $x>0$, 0 when $x=0$, and -1 if $x<0$, e.g., $\text{Sign}(2.2)=1$, $\text{Sign}(0)=0$, $\text{Sign}(-3.5)=-1$.

Sqrt(x)

Returns the square root of x , which has to be non-negative, e.g., $\text{Sqrt}(2)=1.41421$.

SqrtPi(x)

Returns square root of π multiplied by x , which has to be non-negative, e.g. $\text{SqrtPi}(2)=\text{Sqrt}(\text{Pi}()*2)=2.50663$. This function is provided for the sake of compatibility with Microsoft Excel.

Sum(x1,x2,...)

Returns the sum of its arguments, e.g., $\text{Sum}(2.2, 3.5, 1.3)=7.0$. $\text{Sum}()$ requires at least two arguments.

SumSq(x1,x2,...)

Returns the sum of squares of its arguments, e.g., $\text{SumSq}(2.2, 3.5, 1.3)=18.78$. $\text{SumSq}()$ requires at least two arguments.

Trim(x,lo,hi)

Trims the value of the argument x to a value in the interval $\langle lo, hi \rangle$. If $x \leq lo$, the function returns lo , if $x \geq hi$, the function returns hi , if $lo < x < hi$, the function returns x . The function is a shortcut to two nested conditional functions $\text{If}()$ and is equivalent to $\text{If}(x < lo, lo, \text{If}(x > hi, hi, x))$. For example, $\text{Trim}(-0.5, 0, 1)=0$, $\text{Trim}(0.5, 0, 1)=0.5$, $\text{Trim}(1.5, 0, 1)=1$.

Truncate(x)

Returns the integer part of x , e.g., $\text{Truncate}(2.2)=2$.

8.5 Combinatoric Functions

Combin(n,k)

Returns the number of combinations of distinct k elements from among n elements, e.g., $\text{Combin}(10, 2)=45$.

Fact(n)

Returns the factorial of n , e.g., $\text{Fact}(5)=5!=120$. Fact of a negative number returns 0.

FactDouble(n)

Returns the product of all even (when n is even) or all odd (when n is odd) numbers between 1 and n , e.g., $\text{FactDouble}(5)=15$, $\text{FactDouble}(6)=48$. FactDouble of a negative number returns 0.

Multinomial(n1,n2,...)

Factorial of sum of arguments, divided by the factorials of all arguments, e.g., $\text{Multinomial}(2, 5, 3) = \text{Fact}(2+5+3) / (\text{Fact}(2) * \text{Fact}(5) * \text{Fact}(3)) = 10! / (2! * 5! * 3!) = 2520$. All arguments of Multinomial have to be positive.

8.6 Trigonometric Functions

Acos(x)

Returns arccosine (*arcus cosinus*) of x , e.g., $\text{Acos}(-1)=3.14159$.

Asin(x)

Returns arcsine (*arcus sinus*) of x , e.g., $\text{Asin}(1)=1.5708$.

Atan(x)

Returns arctangent (*arcus tangens*) of x , e.g., $\text{Atan}(1)=0.785398$.

Atan2(y,x)

Returns arctangent (*arcus tangens*) from x and y coordinates, e.g., $\text{Atan2}(1,1)=0.785398$.

Cos(x)

Returns cosine (*cosinus*) of x , e.g., $\text{Cos}(1)=0.540302$.

Pi()

Returns constant π , e.g., $\text{Pi}()=3.14159$.

Sin(x)

Returns sine (*sinus*) of x , e.g., $\text{Sin}(1)=0.841471$.

Tan(x)

Returns tangent (*tangens*) of x , e.g., $\text{Tan}(1)=1.55741$.

8.7 Hyperbolic Functions

Cosh(x)

Returns the hyperbolic cosine of x , e.g., $\text{Cosh}(1)=1.54308$.

Sinh(x)

Returns the hyperbolic sine of x , e.g., $\text{Sinh}(1)=1.1752$.

Tanh(x)

Returns the hyperbolic tangent of x , e.g., $\text{Tanh}(1)=0.761594$.

8.8 Logical/Conditional functions

And(b1,b2,...)

Returns the logical conjunction of the arguments, which are all interpreted as Boolean expressions. If any of the expressions evaluates to a zero, `And` returns 0. For example, $\text{And}(1=1, 2, 3)=1$, $\text{And}(1=2, 2, 3)=0$.

Choose(index,v0,v1,...,vn)

Returns v_i if index is equal to i . When index evaluates to a value smaller than 0 or larger than $n-1$, the function returns 0. Examples:

```
Choose(0,1,2,3,4,5)=1
Choose(4,1,2,3,4,5)=5
Choose(7,1,2,3,4,5)=0
```

If(cond,tval,fval)

If *cond* evaluates to non-zero return *tval*, *fval* otherwise, e.g., If(1=2,3,4)=4, If(1,5,10)=5.

Max(x1,x2,...)

Returns the largest of the arguments x_i . Examples:

```
Max(1,2,3,4,5)=5
Max(-3,2,0,2,1)=2
```

Min(x1,x2,...)

Returns the smallest of the arguments x_i . Examples:

```
Min(1,2,3,4,5)=1
Min(-3,2,0,2,1)=-3
```

Or(b1,b2,...)

Returns the logical disjunction of the arguments, which are all interpreted as Boolean expressions. If any of the expressions evaluates to a non-zero value, Or returns 1. For example, Or(1=1,0,0)=1, Or(1=0,0,0)=0.

Switch(x,a1,b1,a2,b2,...,[def])

If $x=a1$, return $b1$, if $x=a2$, return $b2$, when x is not equal to any of a s, return *def* (default value), which is an optional argument. When x is not equal to any of a s and no *def* is defined, return 0. Examples:

```
Switch(3,1,111,2,222,3,333,4,444,5,555,999)=333
Switch(8,1,111,2,222,3,333,4,444,5,555,999)=999
Switch(8,1,111,2,222,3,333,4,444,5,555)=0
```

Xor(b1,b2,...)

Returns logical exclusive OR of all arguments, which are all interpreted as Boolean expressions. Xor returns a 1 if the number of logical expressions that evaluate to non-zero is odd and a 0 otherwise.

Examples:

```
Xor(0,1,2,-3,0)=1
Xor(1,1,0,2,2)=0
Xor(-5,1,1,-2,2)=1
```

8.9 Custom Functions

It is possible to extend the available function set by calling `Network.setExtFunctions`. The functions are defined at the network level and stored within XDSL file. `setExtFunctions` requires an array of strings with the textual definition of functions. Each function definition has the following form:

```
function-name([parameter1[,parameter2...]]) = expression
```

For example, a function adding two numbers can be defined by the following string:

```
add(a,b)=a+b
```

Recursion is not allowed, but it is possible to use the preceding custom functions in the function expression, for example:

```
dbl(x)=x+x  
quad(x)=2*dbl(x)
```

Reverse ordering of `dbl` and `quad` in the vector passed to `setExtFunctions` will result in a parser error for `quad`, because the `dbl` function is not defined when `quad` is parsed.

Each parameter specified on the LHS of the equal sign must be used in the function body on the RHS. The following is an incorrect custom function definition, because parameter `c` is not used:

```
myFunc(a,b,c)=sin(a)+cos(b)
```

Custom function can be defined as a constant. In such case the function has no parameters, but its definition and function calls in node equations still require parenthesis:

```
gEarth()=9.80665
```

Random number generators can be used in function definitions:

```
StdNormal()=Normal(0,1)  
TwoStep(lo,hi)=Steps(lo,(lo+hi)/2,hi,1,2)
```

To retrieve the custom functions defined for the network, call `Network.get_ext_functions`. By default, there are no custom functions. Each call to `set_ext_functions` sets up a complete set of custom functions. To add new functions to the existing set, call `get_ext_functions` first, then add new function definition to the output vector before calling `set_ext_functions`.

This page is intentionally left blank.

PySMILE reference

9 PySMILE reference

Subsections of this topic contain autogenerated reference documentation for classes defined in PySMILE.

9.1 `pysmile.AnnealedMapResults`

map_states

Most likely states of specified map nodes

prob_e

$P(e)$

prob_m1e

$P(m|e)$

9.2 `pysmile.AnnealedMapTuning`

i_reheat_steps

Number of reheating steps in annealing

i_stop_steps

Number of stopping steps

k_map

Scaling parameter for MAP

k_reheat

Scaling factor for reheating

k_rfc

Reheating frequency control parameter

num_cycle

Number of annealing cycles

rand_seed

Random seed used in annealing

speed

Annealing speed

t_init

Initial temperature

t_min

Minimum temperature

9.3 **pysmile.BayesianAlgorithmType**

LAURITZEN

Lauritzen's exact inference algorithm

HENRION

Henrion's algorithm

PEARL

Pearl's exact algorithm

L_SAMPLING

Likelihood sampling

SELF_IMPORTANCE

Self importance sampling

HEURISTIC_IMPORTANCE

Heuristic importance sampling

BACK_SAMPLING

Backward sampling algorithm

AIS_SAMPLING

Adaptive Importance Sampling algorithm

EPIS_SAMPLING

Evidence Pre-propagation Importance Sampling

LBP

Loopy Belief Propagation algorithm

9.4 **pysmile.CaseEvidenceInfo**

continuous

Continuous evidence value

discrete

Discrete evidence value

slice

Time slice index for temporal evidence

type

Type of evidence (discrete, continuous, or virtual)

virtual

Virtual evidence vector (likelihoods over outcomes)

9.5 `pysmile.CaseEvidenceType`

DISCRETE

Discrete evidence

CONTINUOUS

Continuous evidence

VIRTUAL

Virtual evidence

9.6 `pysmile.DeMorganParentType`

INHIBITOR

Inhibitor logic parent

REQUIREMENT

Requirement logic parent

CAUSE

Causal parent

BARRIER

Barrier logic parent

9.7 `pysmile.DiagNetwork`

`find_most_likely_fault(self: pysmile.DiagNetwork) -> int`

Finds the most likely fault

`get_d_sep(self: pysmile.DiagNetwork) -> bool`

Gets d-separation state

`get_entropy_cost_ratio(self: pysmile.DiagNetwork) -> float`

Gets entropy/cost ratio

`get_fault(self: pysmile.DiagNetwork, fault_index: int) -> pysmile.FaultInfo`

Gets specific fault by index

`get_fault_count(self: pysmile.DiagNetwork) -> int`

Gets total number of faults

`get_fault_index(self: pysmile.DiagNetwork, node_handle: int, outcome_index: int) -> int`

`get_fault_index(self: pysmile.DiagNetwork, node_handle: int, outcome_id: str) -> int`

`get_fault_index(self: pysmile.DiagNetwork, node_id: str, outcome_id: str) -> int`

`get_fault_index(self: pysmile.DiagNetwork, node_id: str, outcome_index: int) -> int`

Gets index of a fault

`get_fault_node(self: pysmile.DiagNetwork, fault_index: int) -> int`

Gets fault node object

get_fault_node_id(self: pysmile.DiagNetwork, fault_index: int) -> str

Gets fault node ID

get_fault_outcome(self: pysmile.DiagNetwork, fault_index: int) -> int

Gets fault outcome

get_fault_outcome_id(self: pysmile.DiagNetwork, fault_index: int) -> str

Gets fault outcome ID

get_max_entropy_cost_ratio(self: pysmile.DiagNetwork) -> float

Gets maximum entropy/cost ratio

get_multi_fault_algorithm(self: pysmile.DiagNetwork) -> pysmile.MultiFaultAlgorithmType

Gets current multi-fault reasoning algorithm

get_pursued_fault(self: pysmile.DiagNetwork) -> int

Gets pursued fault

get_pursued_faults(self: pysmile.DiagNetwork) -> List[int]

Gets list of pursued faults

get_single_fault_algorithm(self: pysmile.DiagNetwork) -> pysmile.SingleFaultAlgorithmType

Gets current reasoning algorithm

get_unperformed_observation_ids(self: pysmile.DiagNetwork) -> List[str]

Gets IDs of unperformed observations

get_unperformed_observations(self: pysmile.DiagNetwork) -> List[int]

Gets list of unperformed observations

get_unperformed_test_count(self: pysmile.DiagNetwork) -> int

Gets count of unperformed tests

instantiate_observation(self: pysmile.DiagNetwork, node_handle: int, outcome_index: int) -> None

instantiate_observation(self: pysmile.DiagNetwork, node_handle: int, outcome_id: str) -> None

instantiate_observation(self: pysmile.DiagNetwork, node_id: str, outcome_id: str) -> None

instantiate_observation(self: pysmile.DiagNetwork, node_id: str, outcome_index: int) -> None

Instantiates an observation

is_detailed_entropy_enabled(self: pysmile.DiagNetwork) -> bool

Checks if detailed entropy mode is enabled

is_past_observations_measure_enabled(self: pysmile.DiagNetwork) -> bool

Checks whether past observations field will be filled during update

mandatories_instantiated(self: pysmile.DiagNetwork) -> bool

Checks if all mandatory observations are instantiated

release_observation(self: pysmile.DiagNetwork, node_handle: int) -> None

release_observation(self: pysmile.DiagNetwork, node_id: str) -> None

Releases an observation

restart(self: pysmile.DiagNetwork) -> None

Restarts diagnostic inference

set_d_sep(self: pysmile.DiagNetwork, value: bool) -> None

Sets d-separation state

set_detailed_entropy_enabled(self: pysmile.DiagNetwork, enable: bool) -> None

Enables or disables detailed entropy

set_entropy_cost_ratio(self: pysmile.DiagNetwork, entropy_cost_ratio: float) -> None

Sets entropy/cost ratio

set_multi_fault_algorithm(self: pysmile.DiagNetwork, type: pysmile.MultiFaultAlgorithmType) -> None

Sets multi-fault reasoning algorithm

set_past_observations_measure_enabled(self: pysmile.DiagNetwork, enable: bool) -> None

Enables or disables filling of past observations field during update

set_pursued_fault(self: pysmile.DiagNetwork, fault_index: int) -> None

Sets a pursued fault

set_pursued_faults(self: pysmile.DiagNetwork, fault_indices: List[int]) -> None

Sets list of pursued faults

set_single_fault_algorithm(self: pysmile.DiagNetwork, type: pysmile.SingleFaultAlgorithmType) -> None

Gets current reasoning algorithm

update(self: pysmile.DiagNetwork) -> pysmile.DiagResults

Updates the diagnostic reasoning state

update_fault_beliefs(self: pysmile.DiagNetwork) -> None

Updates beliefs for fault nodes

9.8 **pysmile.DiagResults**

faults

List of diagnosed faults

observations

List of observations

past_observations

List of past observations (if enabled)

9.9 **pysmile.DiscretizationInterval**

boundary

Interval boundary value

id

Interval identifier

9.10 **pysmile.DocItemInfo**

path

Document path

title

Document title

9.11 **pysmile.EPISParams**

num_state_big

Maximum number of states considered 'big'

num_state_medium

Maximum number of states considered 'medium'

num_state_small

Maximum number of states considered 'small'

propagation_length

Number of steps for evidence propagation (sampling depth)

state_big_epsilon

Epsilon threshold for big state groups

state_medium_epsilon

Epsilon threshold for medium state groups

state_otherwise

Epsilon threshold for all other state groups

state_small_epsilon

Epsilon threshold for small state groups

9.12 **pysmile.FaultInfo**

index

Index of the fault

is_pursued

Whether the fault is pursued

node

Associated fault node

outcome

Outcome of the fault

probability

Probability of the fault

9.13 **pysmile.InfluenceDiagramAlgorithmType**

POLICY_EVALUATION

Influence diagram policy evaluation

FIND_BEST_POLICY

Policy search algorithm

9.14 **pysmile.License**

9.15 **pysmile.MpeResults**

mpe_nodes

List of nodes included in the MPE result

mpe_states

Most probable states assigned to the nodes

prob_e

$P(e)$

prob_m1e

$P(m|e)$

9.16 **pysmile.MultiFaultAlgorithmType**

INDEPENDENCE_AT_LEAST_ONE

Independence: at least one fault

INDEPENDENCE_ONLY_ONE

Independence: only one fault

INDEPENDENCE_ONLY_ALL

Independence: only all faults together

DEPENDENCE_AT_LEAST_ONE

Dependence: at least one fault

DEPENDENCE_ONLY_ONE

Dependence: only one fault

DEPENDENCE_ONLY_ALL

Dependence: only all faults together

MARGINAL_1

Marginal model #1

MARGINAL_2

Marginal model #2

MAX_PROB_CHANGE

Maximum probability change

L2_NORMALIZED_DISTANCE

L2 normalized distance

COSINE_DISTANCE

Cosine distance

CITYBLOCK_DISTANCE

Cityblock (Manhattan) distance

AVG_L2_CITY_DISTANCE

Average of L2 and Cityblock distances

9.17 **pysmile.Network**

add_arc(self: pysmile.Network, parent_handle: int, child_handle: int) -> None

add_arc(self: pysmile.Network, parent_id: str, child_id: str) -> None

Adds a directed arc between two nodes

add_case(self: pysmile.Network, case_name: str) -> int

Adds a new case to the network

add_cost_arc(self: pysmile.Network, parent_handle: int, child_handle: int) -> None

add_cost_arc(self: pysmile.Network, parent_id: str, child_id: str) -> None

Adds a cost arc between two nodes

add_node(self: pysmile.Network, node_type: pysmile.NodeType, node_id: str) -> int

add_node(self: pysmile.Network, node_type: pysmile.NodeType) -> int

add_node(self: pysmile.Network) -> int

Adds a node to the network

add_outcome(self: pysmile.Network, node_handle: int, outcome_id: str) -> None

add_outcome(self: pysmile.Network, node_id: str, outcome_id: str) -> None

Adds an outcome to a node

add_submodel(self: pysmile.Network, parent_handle: int, id: str) -> int

add_submodel(self: pysmile.Network, parent_id: str, id: str) -> int

Adds a submodel to the network

add_temporal_arc(self: pysmile.Network, parent_handle: int, child_handle: int, order: int) -> None

add_temporal_arc(self: pysmile.Network, parent_id: str, child_id: str, order: int) -> None

Adds a temporal arc

annealed_map(self: pysmile.Network, map_nodes: List[int], tuning: pysmile.AnnealedMapTuning) ->

pysmile.AnnealedMapResults

annealed_map(self: pysmile.Network, map_nodes: List[str], tuning: pysmile.AnnealedMapTuning) ->

pysmile.AnnealedMapResults

annealed_map(self: pysmile.Network, map_nodes: List[int]) -> pysmile.AnnealedMapResults

annealed_map(self: pysmile.Network, map_nodes: List[str]) -> pysmile.AnnealedMapResults

Performs MAP inference using simulated annealing

```
apply_case_to_network(self: pysmile.Network, case_index: int) -> None
apply_case_to_network(self: pysmile.Network, case_name: str) -> None
```

Applies case evidence to the current network state

```
calc_sensitivity(self: pysmile.Network) -> pysmile.SensitivityResults
```

Performs sensitivity analysis for a node

```
clear_all_evidence(self: pysmile.Network) -> None
```

Clears all evidence in the network

```
clear_all_targets(self: pysmile.Network) -> None
```

Clears all target flags in the network

```
clear_controlled_value(self: pysmile.Network, node_handle: int) -> None
clear_controlled_value(self: pysmile.Network, node_id: str) -> None
```

Clears a node controlled value

```
clear_evidence(self: pysmile.Network, node_handle: int) -> None
clear_evidence(self: pysmile.Network, node_id: str) -> None
```

Clears evidence from a node

```
clear_temporal_evidence(self: pysmile.Network, node_handle: int, slice: int) -> None
clear_temporal_evidence(self: pysmile.Network, node_id: str, slice: int) -> None
```

Clears temporal evidence in a node

```
delete_all_cases(self: pysmile.Network) -> None
```

Deletes all stored cases

```
delete_arc(self: pysmile.Network, parent_handle: int, child_handle: int) -> None
delete_arc(self: pysmile.Network, parent_id: str, child_id: str) -> None
```

Removes an arc between two nodes

```
delete_case(self: pysmile.Network, case_index: int) -> None
delete_case(self: pysmile.Network, case_name: str) -> None
```

Deletes a case

```
delete_case_evidence(self: pysmile.Network, case_index: int, node_handle: int) -> None
delete_case_evidence(self: pysmile.Network, case_index: int, node_id: str) -> None
delete_case_evidence(self: pysmile.Network, case_name: str, node_handle: int) -> None
delete_case_evidence(self: pysmile.Network, case_name: str, node_id: str) -> None
```

Deletes all evidence for a case

```
delete_cost_arc(self: pysmile.Network, parent_handle: int, child_handle: int) -> None
delete_cost_arc(self: pysmile.Network, parent_id: str, child_id: str) -> None
```

Removes a cost arc between nodes

```
delete_node(self: pysmile.Network, node_handle: int) -> None
delete_node(self: pysmile.Network, node_id: str) -> None
```

Deletes a node

```
delete_outcome(self: pysmile.Network, node_handle: int, outcome_id: str) -> None
delete_outcome(self: pysmile.Network, node_id: str, outcome_id: str) -> None
```

```
delete_outcome(self: pysmile.Network, node_handle: int, outcome_index: int) -> None
delete_outcome(self: pysmile.Network, node_id: str, outcome_index: int) -> None
```

Deletes an outcome from a node

```
delete_submodel(self: pysmile.Network, handle: int) -> None
delete_submodel(self: pysmile.Network, id: str) -> None
```

Deletes a submodel

```
delete_temporal_arc(self: pysmile.Network, parent_handle: int, child_handle: int, order: int) -> None
delete_temporal_arc(self: pysmile.Network, parent_id: str, child_id: str, order: int) -> None
```

Deletes a temporal arc

```
distributed_helper_em(self: pysmile.Network) -> List[float]
```

Creates distributed EM helper

```
enable_case_sync(self: pysmile.Network, enable: bool) -> None
```

Enables or disables case-to-network synchronization

```
find_case(self: pysmile.Network, case_name: str) -> int
```

Finds the index of a case by name

```
get_all_case_names(self: pysmile.Network) -> List[str]
```

Returns a list of all case names

```
get_all_node_ids(self: pysmile.Network) -> List[str]
```

Returns IDs of all nodes

```
get_all_nodes(self: pysmile.Network) -> List[int]
```

Returns handles to all nodes

```
get_bayesian_algorithm(self: pysmile.Network) -> pysmile.BayesianAlgorithmType
```

Returns the Bayesian algorithm used

```
get_case_category(self: pysmile.Network, case_index: int) -> str
```

```
get_case_category(self: pysmile.Network, case_name: str) -> str
```

Returns the category of a case

```
get_case_count(self: pysmile.Network) -> int
```

Returns the total number of cases in the network

```
get_case_created(self: pysmile.Network, case_index: int) -> int
```

```
get_case_created(self: pysmile.Network, case_name: str) -> int
```

Returns the creation timestamp of a case

```
get_case_description(self: pysmile.Network, case_index: int) -> str
```

```
get_case_description(self: pysmile.Network, case_name: str) -> str
```

Returns the description of a case

```
get_case_evidence(self: pysmile.Network, case_index: int, node_handle: int) -> pysmile.CaseEvidenceInfo
```

```
get_case_evidence(self: pysmile.Network, case_index: int, node_id: str) -> pysmile.CaseEvidenceInfo
```

```
get_case_evidence(self: pysmile.Network, case_name: str, node_handle: int) -> pysmile.CaseEvidenceInfo
```

```
get_case_evidence(self: pysmile.Network, case_name: str, node_id: str) -> pysmile.CaseEvidenceInfo
```

Gets evidence from a case

```
get_case_evidence_count(self: pysmile.Network, case_index: int) -> int
get_case_evidence_count(self: pysmile.Network, case_name: str) -> int
```

Returns the number of evidence entries in a case

```
get_case_evidence_node_ids(self: pysmile.Network, case_index: int) -> List[str]
get_case_evidence_node_ids(self: pysmile.Network, case_name: str) -> List[str]
```

Returns IDs of nodes with evidence in a case

```
get_case_evidence_nodes(self: pysmile.Network, case_index: int) -> List[int]
get_case_evidence_nodes(self: pysmile.Network, case_name: str) -> List[int]
```

Returns handles of nodes with evidence in a case

```
get_case_last_modified(self: pysmile.Network, case_index: int) -> int
get_case_last_modified(self: pysmile.Network, case_name: str) -> int
```

Returns the last modified timestamp of a case

```
get_case_target_count(self: pysmile.Network, case_index: int) -> int
get_case_target_count(self: pysmile.Network, case_name: str) -> int
```

Returns the number of target nodes in a case

```
get_case_target_ids(self: pysmile.Network, case_index: int) -> List[str]
get_case_target_ids(self: pysmile.Network, case_name: str) -> List[str]
```

Returns the IDs of target nodes in a case

```
get_case_targets(self: pysmile.Network, case_index: int) -> List[int]
get_case_targets(self: pysmile.Network, case_name: str) -> List[int]
```

Returns handles of target nodes in a case

```
get_case_temporal_evidence(self: pysmile.Network, case_index: int, node_handle: int) ->
List[pysmile.CaseEvidenceInfo]
get_case_temporal_evidence(self: pysmile.Network, case_index: int, node_id: str) ->
List[pysmile.CaseEvidenceInfo]
get_case_temporal_evidence(self: pysmile.Network, case_name: str, node_handle: int) ->
List[pysmile.CaseEvidenceInfo]
get_case_temporal_evidence(self: pysmile.Network, case_name: str, node_id: str) ->
List[pysmile.CaseEvidenceInfo]
```

Gets temporal evidence from a case

```
get_child_ids(self: pysmile.Network, node_handle: int) -> List[str]
get_child_ids(self: pysmile.Network, node_id: str) -> List[str]
```

Returns child node IDs

```
get_children(self: pysmile.Network, node_handle: int) -> List[int]
get_children(self: pysmile.Network, node_id: str) -> List[int]
```

Returns child node handles

```
get_cont_evidence(self: pysmile.Network, node_handle: int) -> float
get_cont_evidence(self: pysmile.Network, node_id: str) -> float
```

Returns continuous evidence for a node

```
get_controlled_value(self: pysmile.Network, node_handle: int) -> int
get_controlled_value(self: pysmile.Network, node_id: str) -> int
```

Returns controlled value of a node

```
get_controlled_value_id(self: pysmile.Network, node_handle: int) -> str
get_controlled_value_id(self: pysmile.Network, node_id: str) -> str
```

Returns ID of the controlled value

```
get_cost_child_ids(self: pysmile.Network, node_handle: int) -> List[str]
get_cost_child_ids(self: pysmile.Network, node_id: str) -> List[str]
```

Returns child node IDs in cost arcs

```
get_cost_children(self: pysmile.Network, node_handle: int) -> List[int]
get_cost_children(self: pysmile.Network, node_id: str) -> List[int]
```

Returns child node handles in cost arcs

```
get_cost_parent_ids(self: pysmile.Network, node_handle: int) -> List[str]
get_cost_parent_ids(self: pysmile.Network, node_id: str) -> List[str]
```

Returns parent node IDs in cost arcs

```
get_cost_parents(self: pysmile.Network, node_handle: int) -> List[int]
get_cost_parents(self: pysmile.Network, node_id: str) -> List[int]
```

Returns parent node handles in cost arcs

```
get_de_morgan_parent_type(self: pysmile.Network, node_handle: int, parent_index: int) ->
pysmile.DeMorganParentType
get_de_morgan_parent_type(self: pysmile.Network, node_id: str, parent_index: int) -> pysmile.DeMorganParentType
get_de_morgan_parent_type(self: pysmile.Network, node_handle: int, parent_id: str) ->
pysmile.DeMorganParentType
get_de_morgan_parent_type(self: pysmile.Network, node_id: str, parent_id: str) -> pysmile.DeMorganParentType
```

Returns DeMorgan parent type

```
get_de_morgan_parent_weight(self: pysmile.Network, node_handle: int, parent_index: int) -> float
get_de_morgan_parent_weight(self: pysmile.Network, node_id: str, parent_index: int) -> float
get_de_morgan_parent_weight(self: pysmile.Network, node_handle: int, parent_id: str) -> float
get_de_morgan_parent_weight(self: pysmile.Network, node_id: str, parent_id: str) -> float
```

Returns DeMorgan parent weight

```
get_de_morgan_prior_belief(self: pysmile.Network, node_handle: int) -> float
get_de_morgan_prior_belief(self: pysmile.Network, node_id: str) -> float
```

Returns DeMorgan prior belief

```
get_default_outcome(self: pysmile.Network, node_handle: int) -> int
get_default_outcome(self: pysmile.Network, node_id: str) -> int
```

Returns the default outcome label

```
get_default_outcome_id(self: pysmile.Network, node_handle: int) -> str
get_default_outcome_id(self: pysmile.Network, node_id: str) -> str
```

Returns the ID of the default outcome

```
get_description(self: pysmile.Network) -> str
```

Returns network description

```
get_discretization_sample_count(self: pysmile.Network) -> int
```

Returns sample count used for discretization

```
get_epis_params(self: pysmile.Network) -> pysmile.EPISParams
```

Returns the current EPIS sampling parameters

```
get_evidence(self: pysmile.Network, node_handle: int) -> int
get_evidence(self: pysmile.Network, node_id: str) -> int
```

Returns evidence for a node

```
get_evidence_id(self: pysmile.Network, node_handle: int) -> str  
get_evidence_id(self: pysmile.Network, node_id: str) -> str
```

Returns evidence outcome ID

```
get_ext_functions(self: pysmile.Network) -> List[str]
```

Returns external functions

```
get_first_node(self: pysmile.Network) -> int
```

Returns handle to first node

```
get_first_node_id(self: pysmile.Network) -> str
```

Returns ID of first node

```
get_first_submodel(self: pysmile.Network) -> int
```

Returns handle to first submodel

```
get_id(self: pysmile.Network) -> str
```

Returns network ID

```
get_influence_diagram_algorithm(self: pysmile.Network) -> pysmile.InfluenceDiagramAlgorithmType
```

Returns influence diagram algorithm

```
get_intervals(self: pysmile.Network, node_handle: int) -> List[float]
```

```
get_intervals(self: pysmile.Network, node_id: str) -> List[float]
```

Returns interval values

```
get_library_build(self: pysmile.Network) -> str
```

Returns library build version

```
get_main_submodel(self: pysmile.Network) -> int
```

Returns main submodel handle

```
get_main_submodel_id(self: pysmile.Network) -> str
```

Returns ID of main submodel

```
get_mau_expressions(self: pysmile.Network, node_handle: int) -> List[str]
```

```
get_mau_expressions(self: pysmile.Network, node_id: str) -> List[str]
```

Returns MAU expressions

```
get_max_node_temporal_order(self: pysmile.Network, node_handle: int) -> int
```

```
get_max_node_temporal_order(self: pysmile.Network, node_id: str) -> int
```

Returns max temporal order of a node

```
get_max_temporal_order(self: pysmile.Network) -> int
```

Returns max temporal order in the network

```
get_name(self: pysmile.Network) -> str
```

Returns network name

```
get_next_node(self: pysmile.Network, node_handle: int) -> int
```

```
get_next_node(self: pysmile.Network, node_id: str) -> int
```

Returns next node handle

```
get_next_node_id(self: pysmile.Network, node_handle: int) -> str  
get_next_node_id(self: pysmile.Network, node_id: str) -> str
```

Returns next node ID

```
get_next_submodel(self: pysmile.Network, handle: int) -> int
```

Returns next submodel handle

```
get_node(self: pysmile.Network, node_id: str) -> int
```

Returns node handle

```
get_node_bg_color(self: pysmile.Network, node_handle: int) -> int  
get_node_bg_color(self: pysmile.Network, node_id: str) -> int
```

Returns background color of a node

```
get_node_border_color(self: pysmile.Network, node_handle: int) -> int  
get_node_border_color(self: pysmile.Network, node_id: str) -> int
```

Returns border color of a node

```
get_node_border_width(self: pysmile.Network, node_handle: int) -> int  
get_node_border_width(self: pysmile.Network, node_id: str) -> int
```

Returns node border width

```
get_node_chart_size(self: pysmile.Network, node_handle: int) -> List[int]  
get_node_chart_size(self: pysmile.Network, node_id: str) -> List[int]
```

Returns node chart size

```
get_node_cost(self: pysmile.Network, node_handle: int) -> List[float]  
get_node_cost(self: pysmile.Network, node_id: str) -> List[float]
```

Returns cost value of node

```
get_node_count(self: pysmile.Network) -> int
```

Returns number of nodes in network

```
get_node_definition(self: pysmile.Network, node_handle: int) -> List[float]  
get_node_definition(self: pysmile.Network, node_id: str) -> List[float]
```

Returns definition table of node

```
get_node_description(self: pysmile.Network, node_handle: int) -> str  
get_node_description(self: pysmile.Network, node_id: str) -> str
```

Returns description of a node

```
get_node_diag_type(self: pysmile.Network, node_handle: int) -> pysmile.NodeDiagType  
get_node_diag_type(self: pysmile.Network, node_id: str) -> pysmile.NodeDiagType
```

Returns diagnostic type of node

```
get_node_documentation(self: pysmile.Network, node_handle: int) -> List[pysmile.DocItemInfo]  
get_node_documentation(self: pysmile.Network, node_id: str) -> List[pysmile.DocItemInfo]
```

Returns documentation of a node

```
get_node_equation(self: pysmile.Network, node_handle: int) -> str  
get_node_equation(self: pysmile.Network, node_id: str) -> str
```

Returns node equation

```
get_node_equation_bounds(self: pysmile.Network, node_handle: int) -> List[float]  
get_node_equation_bounds(self: pysmile.Network, node_id: str) -> List[float]
```

Returns bounds of the node equation

```
get_node_equation_constant(self: pysmile.Network, node_handle: int) -> float  
get_node_equation_constant(self: pysmile.Network, node_id: str) -> float
```

Returns constant flag for node equation

```
get_node_equation_discretization(self: pysmile.Network, node_handle: int) ->  
List[pysmile.DiscretizationInterval]  
get_node_equation_discretization(self: pysmile.Network, node_id: str) -> List[pysmile.DiscretizationInterval]
```

Returns discretization for node equation

```
get_node_id(self: pysmile.Network, node_handle: int) -> str
```

Returns ID of node

```
get_node_mean(self: pysmile.Network, node_handle: int) -> float  
get_node_mean(self: pysmile.Network, node_id: str) -> float
```

Returns mean value of a continuous node

```
get_node_name(self: pysmile.Network, node_handle: int) -> str  
get_node_name(self: pysmile.Network, node_id: str) -> str
```

Returns node name

```
get_node_position(self: pysmile.Network, node_handle: int) -> List[int]  
get_node_position(self: pysmile.Network, node_id: str) -> List[int]
```

Returns position of node

```
get_node_question(self: pysmile.Network, node_handle: int) -> str  
get_node_question(self: pysmile.Network, node_id: str) -> str
```

Returns question string of node

```
get_node_sample_stats(self: pysmile.Network, node_handle: int) -> List[float]  
get_node_sample_stats(self: pysmile.Network, node_id: str) -> List[float]
```

Returns sample statistics for node

```
get_node_std_dev(self: pysmile.Network, node_handle: int) -> float  
get_node_std_dev(self: pysmile.Network, node_id: str) -> float
```

Returns standard deviation for continuous node

```
get_node_temporal_definition(self: pysmile.Network, node_handle: int, order: int) -> List[float]  
get_node_temporal_definition(self: pysmile.Network, node_id: str, order: int) -> List[float]
```

Returns temporal definition table

```
get_node_temporal_mean(self: pysmile.Network, node_handle: int) -> List[float]  
get_node_temporal_mean(self: pysmile.Network, node_id: str) -> List[float]
```

Returns the temporal mean for the specified node

```
get_node_temporal_std_dev(self: pysmile.Network, node_handle: int) -> List[float]  
get_node_temporal_std_dev(self: pysmile.Network, node_id: str) -> List[float]
```

Returns the temporal standard deviation for the specified node

```
get_node_temporal_type(self: pysmile.Network, node_handle: int) -> pysmile.NodeTemporalType  
get_node_temporal_type(self: pysmile.Network, node_id: str) -> pysmile.NodeTemporalType
```

Returns temporal type of node

```
get_node_text_color(self: pysmile.Network, node_handle: int) -> int
get_node_text_color(self: pysmile.Network, node_id: str) -> int
```

Returns node text color

```
get_node_type(self: pysmile.Network, node_handle: int) -> int
get_node_type(self: pysmile.Network, node_id: str) -> int
```

Returns node type

```
get_node_user_properties(self: pysmile.Network, node_handle: int) -> List[pysmile.UserProperty]
get_node_user_properties(self: pysmile.Network, node_id: str) -> List[pysmile.UserProperty]
```

Returns user-defined properties for node

```
get_node_value(self: pysmile.Network, node_handle: int) -> List[float]
get_node_value(self: pysmile.Network, node_id: str) -> List[float]
```

Returns belief value of node

```
get_noisy_decomposition_limit(self: pysmile.Network) -> int
```

Returns limit on noisy decomposition

```
get_noisy_expanded_definition(self: pysmile.Network, node_handle: int) -> List[float]
get_noisy_expanded_definition(self: pysmile.Network, node_id: str) -> List[float]
```

Returns expanded noisy-OR definition

```
get_noisy_parent_strengths(self: pysmile.Network, node_handle: int, parent_index: int) -> List[int]
get_noisy_parent_strengths(self: pysmile.Network, node_id: str, parent_index: int) -> List[int]
get_noisy_parent_strengths(self: pysmile.Network, node_handle: int, parent_id: str) -> List[int]
get_noisy_parent_strengths(self: pysmile.Network, node_id: str, parent_id: str) -> List[int]
```

Returns noisy parent strengths

```
get_noisy_temporal_parent_strengths(self: pysmile.Network, node_handle: int, order: int) -> List[List[int]]
get_noisy_temporal_parent_strengths(self: pysmile.Network, node_id: str, order: int) -> List[List[int]]
```

Returns noisy temporal parent strengths

```
get_outcome_count(self: pysmile.Network, node_handle: int) -> int
get_outcome_count(self: pysmile.Network, node_id: str) -> int
```

Returns number of outcomes for a node

```
get_outcome_description(self: pysmile.Network, node_handle: int, outcome_index: int) -> str
get_outcome_description(self: pysmile.Network, node_id: str, outcome_index: int) -> str
get_outcome_description(self: pysmile.Network, node_handle: int, outcome_id: str) -> str
get_outcome_description(self: pysmile.Network, node_id: str, outcome_id: str) -> str
```

Returns description for an outcome

```
get_outcome_documentation(self: pysmile.Network, node_handle: int, outcome_index: int) ->
List[pysmile.DocItemInfo]
get_outcome_documentation(self: pysmile.Network, node_id: str, outcome_index: int) -> List[pysmile.DocItemInfo]
get_outcome_documentation(self: pysmile.Network, node_id: str, outcome_id: str) -> List[pysmile.DocItemInfo]
get_outcome_documentation(self: pysmile.Network, node_handle: int, outcome_id: str) ->
List[pysmile.DocItemInfo]
```

Returns documentation for an outcome

```
get_outcome_fix(self: pysmile.Network, node_handle: int, outcome_index: int) -> str
get_outcome_fix(self: pysmile.Network, node_id: str, outcome_index: int) -> str
get_outcome_fix(self: pysmile.Network, node_handle: int, outcome_id: str) -> str
get_outcome_fix(self: pysmile.Network, node_id: str, outcome_id: str) -> str
```

Returns fix flag for an outcome

```
get_outcome_id(self: pysmile.Network, node_handle: int, outcome_index: int) -> str
get_outcome_id(self: pysmile.Network, node_id: str, outcome_index: int) -> str
```

Returns ID of outcome

```
get_outcome_ids(self: pysmile.Network, arg0: int) -> List[str]
get_outcome_ids(self: pysmile.Network, arg0: str) -> List[str]
```

Returns all outcome IDs

```
get_outcome_label(self: pysmile.Network, node_handle: int, outcome_index: int) -> str
get_outcome_label(self: pysmile.Network, node_id: str, outcome_index: int) -> str
get_outcome_label(self: pysmile.Network, node_handle: int, outcome_id: str) -> str
get_outcome_label(self: pysmile.Network, node_id: str, outcome_id: str) -> str
```

Returns label of an outcome

```
get_parent_ids(self: pysmile.Network, node_handle: int) -> List[str]
get_parent_ids(self: pysmile.Network, node_id: str) -> List[str]
```

Returns parent node IDs

```
get_parents(self: pysmile.Network, node_handle: int) -> List[int]
get_parents(self: pysmile.Network, node_id: str) -> List[int]
```

Returns parent node handles

```
get_point_values(self: pysmile.Network, node_handle: int) -> List[float]
get_point_values(self: pysmile.Network, node_id: str) -> List[float]
```

Returns node point values

```
get_rand_seed(self: pysmile.Network) -> int
```

Returns the current random seed

```
get_sample_count(self: pysmile.Network) -> int
```

Returns number of samples for inference

```
get_slice_count(self: pysmile.Network) -> int
```

Returns number of time slices

```
get_submodel(self: pysmile.Network, id: str) -> int
```

Returns handle to submodel

```
get_submodel_count(self: pysmile.Network) -> int
```

Returns number of submodels

```
get_submodel_description(self: pysmile.Network, handle: int) -> str
get_submodel_description(self: pysmile.Network, id: str) -> str
```

Returns description of submodel

```
get_submodel_id(self: pysmile.Network, handle: int) -> str
```

Returns ID of submodel

```
get_submodel_name(self: pysmile.Network, handle: int) -> str
get_submodel_name(self: pysmile.Network, id: str) -> str
```

Returns name of submodel

```
get_submodel_of_node(self: pysmile.Network, node_handle: int) -> int
get_submodel_of_node(self: pysmile.Network, node_id: str) -> int
```

Returns submodel handle containing a given node

```
get_submodel_of_submodel(self: pysmile.Network, submodel_handle: int) -> int
get_submodel_of_submodel(self: pysmile.Network, submodel_id: str) -> int
```

Returns parent submodel

```
get_submodel_position(self: pysmile.Network, submodel_handle: int) -> List[int]
get_submodel_position(self: pysmile.Network, submodel_id: str) -> List[int]
```

Returns position of submodel

```
get_temporal_children(self: pysmile.Network, node_handle: int) -> List[pysmile.TemporalInfo]
get_temporal_children(self: pysmile.Network, node_id: str) -> List[pysmile.TemporalInfo]
```

Returns temporal child nodes

```
get_temporal_cont_evidence(self: pysmile.Network, node_handle: int, slice: int) -> float
get_temporal_cont_evidence(self: pysmile.Network, node_id: str, slice: int) -> float
```

Returns temporal continuous evidence values

```
get_temporal_evidence(self: pysmile.Network, node_handle: int, slice: int) -> int
get_temporal_evidence(self: pysmile.Network, node_id: str, slice: int) -> int
```

Returns temporal evidence values

```
get_temporal_evidence_id(self: pysmile.Network, node_handle: int, slice: int) -> str
get_temporal_evidence_id(self: pysmile.Network, node_id: str, slice: int) -> str
```

Returns ID of temporal evidence

```
get_temporal_orders(self: pysmile.Network, node_handle: int) -> List[int]
get_temporal_orders(self: pysmile.Network, node_id: str) -> List[int]
```

Returns temporal orders

```
get_temporal_parents(self: pysmile.Network, node_handle: int, order: int) -> List[pysmile.TemporalInfo]
get_temporal_parents(self: pysmile.Network, node_id: str, order: int) -> List[pysmile.TemporalInfo]
```

Returns temporal parent nodes

```
get_temporal_virtual_evidence(self: pysmile.Network, node_handle: int, slice: int) -> List[float]
get_temporal_virtual_evidence(self: pysmile.Network, node_id: str, slice: int) -> List[float]
```

Returns temporal virtual evidence

```
get_unrolled_parents(self: pysmile.Network, node_handle: int, order: int) -> List[pysmile.TemporalInfo]
get_unrolled_parents(self: pysmile.Network, node_id: str, order: int) -> List[pysmile.TemporalInfo]
get_unrolled_parents(self: pysmile.Network, node_handle: int) -> List[pysmile.TemporalInfo]
get_unrolled_parents(self: pysmile.Network, node_id: str) -> List[pysmile.TemporalInfo]
```

Returns unrolled parent nodes

```
get_user_properties(self: pysmile.Network) -> List[pysmile.UserProperty]
```

Returns user-defined network properties

```
get_value_indexing_parent_ids(self: pysmile.Network, node_handle: int) -> List[str]
get_value_indexing_parent_ids(self: pysmile.Network, node_id: str) -> List[str]
```

Returns parent IDs for value indexing

```
get_value_indexing_parents(self: pysmile.Network, node_handle: int) -> List[int]
get_value_indexing_parents(self: pysmile.Network, node_id: str) -> List[int]
```

Returns parent handles for value indexing

```
get_virtual_evidence(self: pysmile.Network, node_handle: int) -> List[float]
get_virtual_evidence(self: pysmile.Network, node_id: str) -> List[float]
```

Returns virtual evidence for node

```
has_intervals(self: pysmile.Network, node_handle: int) -> bool
has_intervals(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node has intervals

```
has_outcome_ids(self: pysmile.Network, node_handle: int) -> bool
has_outcome_ids(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node uses outcome IDs

```
has_point_values(self: pysmile.Network, node_handle: int) -> bool
has_point_values(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node has point values

```
has_temporal_evidence(self: pysmile.Network, node_handle: int) -> bool
has_temporal_evidence(self: pysmile.Network, node_id: str) -> bool
```

Returns true if temporal evidence exists

```
insert_outcome(self: pysmile.Network, node_handle: int, outcome_index: int, outcome_id: str) -> None
insert_outcome(self: pysmile.Network, node_id: str, outcome_index: int, outcome_id: str) -> None
```

Inserts a new outcome to a node

```
is_case_sync_enabled(self: pysmile.Network) -> bool
```

Checks whether case synchronization is enabled

```
is_case_target(self: pysmile.Network, case_index: int, node_handle: int) -> bool
is_case_target(self: pysmile.Network, case_name: str, node_handle: int) -> bool
is_case_target(self: pysmile.Network, case_index: int, node_id: str) -> bool
is_case_target(self: pysmile.Network, case_name: str, node_id: str) -> bool
```

Checks whether a node is a target in the given case

```
is_controlled(self: pysmile.Network, node_handle: int) -> bool
is_controlled(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node is controlled

```
is_evidence(self: pysmile.Network, node_handle: int) -> bool
is_evidence(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node has evidence set

```
is_fault_outcome(self: pysmile.Network, node_handle: int, outcome_index: int) -> bool
is_fault_outcome(self: pysmile.Network, node_id: str, outcome_index: int) -> bool
is_fault_outcome(self: pysmile.Network, node_handle: int, outcome_id: str) -> bool
is_fault_outcome(self: pysmile.Network, node_id: str, outcome_id: str) -> bool
```

Returns true if outcome is fault

```
is_mandatory(self: pysmile.Network, node_handle: int) -> bool
is_mandatory(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node is mandatory

```
is_node_chart_enabled(self: pysmile.Network, node_handle: int) -> bool
is_node_chart_enabled(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node chart is enabled

```
is_node_equation_constant(self: pysmile.Network, node_handle: int) -> bool  
is_node_equation_constant(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node equation is constant

```
is_node_equation_deterministic(self: pysmile.Network, node_handle: int) -> bool  
is_node_equation_deterministic(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node equation is deterministic

```
is_noisy_decomposition_enabled(self: pysmile.Network) -> bool
```

Returns true if noisy decomposition is enabled

```
is_noisy_expanded(self: pysmile.Network, node_handle: int) -> bool  
is_noisy_expanded(self: pysmile.Network, node_id: str) -> bool
```

Returns true if noisy OR is expanded

```
is_noisy_synchronized(self: pysmile.Network, node_handle: int) -> bool  
is_noisy_synchronized(self: pysmile.Network, node_id: str) -> bool
```

Returns true if noisy strengths are synchronized

```
is_outlier_rejection_enabled(self: pysmile.Network) -> bool
```

Returns true if outlier rejection is enabled

```
is_propagated_evidence(self: pysmile.Network, node_handle: int) -> bool  
is_propagated_evidence(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node has propagated evidence

```
is_ranked(self: pysmile.Network, node_handle: int) -> bool  
is_ranked(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node is ranked

```
is_real_evidence(self: pysmile.Network, node_handle: int) -> bool  
is_real_evidence(self: pysmile.Network, node_id: str) -> bool
```

Returns true if real evidence is set

```
is_relevance_enabled(self: pysmile.Network) -> bool
```

Returns true if relevance is enabled

```
is_target(self: pysmile.Network, node_handle: int) -> bool  
is_target(self: pysmile.Network, node_id: str) -> bool
```

Returns true if node is a target

```
is_temporal_cont_evidence(self: pysmile.Network, node_handle: int, slice: int) -> bool  
is_temporal_cont_evidence(self: pysmile.Network, node_id: str, slice: int) -> bool
```

Returns true if node has temporal continuous evidence

```
is_temporal_evidence(self: pysmile.Network, node_handle: int, slice: int) -> bool  
is_temporal_evidence(self: pysmile.Network, node_id: str, slice: int) -> bool
```

Returns true if node has temporal evidence

```
is_temporal_virtual_evidence(self: pysmile.Network, node_handle: int, slice: int) -> bool  
is_temporal_virtual_evidence(self: pysmile.Network, node_id: str, slice: int) -> bool
```

Returns true if node has temporal virtual evidence

`is_value_discretized(self: pysmile.Network, node_handle: int) -> bool`
`is_value_discretized(self: pysmile.Network, node_id: str) -> bool`

Returns true if node values are discretized

`is_value_valid(self: pysmile.Network, node_handle: int) -> bool`
`is_value_valid(self: pysmile.Network, node_id: str) -> bool`

Returns true if node value is valid

`is_virtual_evidence(self: pysmile.Network, node_handle: int) -> bool`
`is_virtual_evidence(self: pysmile.Network, node_id: str) -> bool`

Returns true if node has virtual evidence

`is_zero_avoidance_enabled(self: pysmile.Network) -> bool`

Returns true if zero-avoidance is enabled

`marginalize_node(self: pysmile.Network, node_handle: int) -> None`
`marginalize_node(self: pysmile.Network, node_id: str) -> None`

Remove node from the network, and modify the definitions of the remaining nodes, so that the joint probability distribution over the remaining variables in the network is unchanged

`most_probable_explanation(self: pysmile.Network) -> pysmile.MpeResults`

Computes the Most Probable Explanation (MPE) for the current evidence

`prob_evidence(self: pysmile.Network) -> float`

Returns probability of evidence

`read_file(self: pysmile.Network, xdsl_file: str) -> None`

Reads network from a file

`read_string(self: pysmile.Network, xdsl_string: str) -> None`

Reads network from a string

`redirect_log(self: pysmile.Network, std_file_descriptor: int) -> None`
`redirect_log(self: pysmile.Network, filename: str) -> None`

Redirects SMILE log output

`remove_intervals(self: pysmile.Network, node_handle: int) -> None`
`remove_intervals(self: pysmile.Network, node_id: str) -> None`

Removes interval values

`remove_point_values(self: pysmile.Network, node_handle: int) -> None`
`remove_point_values(self: pysmile.Network, node_id: str) -> None`

Removes point values

`save_network_to_case(self: pysmile.Network, case_index: int) -> None`
`save_network_to_case(self: pysmile.Network, case_name: str) -> None`

Saves current network state to a case

`set_bayesian_algorithm(self: pysmile.Network, bayesian_algorithm: pysmile.BayesianAlgorithmType) -> None`

Sets the Bayesian inference algorithm

`set_case_category(self: pysmile.Network, case_index: int, category: str) -> None`
`set_case_category(self: pysmile.Network, case_name: str, category: str) -> None`

Sets the category of a case

```
set_case_created(self: pysmile.Network, case_index: int, created: int) -> None
set_case_created(self: pysmile.Network, case_name: str, created: int) -> None
```

Sets the creation timestamp of a case

```
set_case_description(self: pysmile.Network, case_index: int, description: str) -> None
set_case_description(self: pysmile.Network, case_name: str, description: str) -> None
```

Sets the description of a case

```
set_case_evidence(self: pysmile.Network, case_index: int, node_handle: int, case_evidence:
pysmile.CaseEvidenceInfo) -> None
set_case_evidence(self: pysmile.Network, case_index: int, node_id: str, case_evidence:
pysmile.CaseEvidenceInfo) -> None
set_case_evidence(self: pysmile.Network, case_name: str, node_handle: int, case_evidence:
pysmile.CaseEvidenceInfo) -> None
set_case_evidence(self: pysmile.Network, case_name: str, node_id: str, case_evidence: pysmile.CaseEvidenceInfo)
-> None
```

Sets evidence for a node in a case

```
set_case_last_modified(self: pysmile.Network, case_index: int, created: int) -> None
set_case_last_modified(self: pysmile.Network, case_name: str, created: int) -> None
```

Sets the last modified timestamp of a case

```
set_case_name(self: pysmile.Network, case_index: int, name: str) -> None
set_case_name(self: pysmile.Network, case_name: str, name: str) -> None
```

Sets the name of a case

```
set_case_targets(self: pysmile.Network, case_index: int, target_handles: List[int]) -> None
set_case_targets(self: pysmile.Network, case_name: str, target_handles: List[int]) -> None
set_case_targets(self: pysmile.Network, case_index: int, target_ids: List[str]) -> None
set_case_targets(self: pysmile.Network, case_name: str, target_ids: List[str]) -> None
```

Sets target nodes for a case

```
set_case_temporal_evidence(self: pysmile.Network, case_index: int, node_handle: int, case_temporal_evidence:
List[pysmile.CaseEvidenceInfo]) -> None
set_case_temporal_evidence(self: pysmile.Network, case_index: int, node_id: str, case_temporal_evidence:
List[pysmile.CaseEvidenceInfo]) -> None
set_case_temporal_evidence(self: pysmile.Network, case_name: str, node_handle: int, case_temporal_evidence:
List[pysmile.CaseEvidenceInfo]) -> None
set_case_temporal_evidence(self: pysmile.Network, case_name: str, node_id: str, case_temporal_evidence:
List[pysmile.CaseEvidenceInfo]) -> None
```

Sets temporal evidence for a node in a case

```
set_cont_evidence(self: pysmile.Network, node_handle: int, evidence: float) -> None
set_cont_evidence(self: pysmile.Network, node_id: str, evidence: float) -> None
```

Sets continuous evidence

```
set_controlled_value(self: pysmile.Network, node_handle: int, outcome_index: int) -> None
set_controlled_value(self: pysmile.Network, node_id: str, outcome_index: int) -> None
set_controlled_value(self: pysmile.Network, node_handle: int, outcome_id: str) -> None
set_controlled_value(self: pysmile.Network, node_id: str, outcome_id: str) -> None
```

Sets controlled value for decision node

```
set_de_morgan_parent_type(self: pysmile.Network, node_handle: int, parent_index: int, parent_type:
pysmile.DeMorganParentType) -> None
set_de_morgan_parent_type(self: pysmile.Network, node_id: str, parent_index: int, parent_type:
pysmile.DeMorganParentType) -> None
set_de_morgan_parent_type(self: pysmile.Network, node_handle: int, parent_id: str, parent_type:
```

```
pysmile.DeMorganParentType) -> None
```

```
set_de_morgan_parent_type(self: pysmile.Network, node_id: str, parent_id: str, parent_type:
```

```
pysmile.DeMorganParentType) -> None
```

Sets DeMorgan parent type

```
set_de_morgan_parent_weight(self: pysmile.Network, node_handle: int, parent_index: int, parent_weight: float) -> None
```

```
set_de_morgan_parent_weight(self: pysmile.Network, node_id: str, parent_index: int, parent_weight: float) -> None
```

```
set_de_morgan_parent_weight(self: pysmile.Network, node_handle: int, parent_id: str, parent_weight: float) -> None
```

```
set_de_morgan_parent_weight(self: pysmile.Network, node_id: str, parent_id: str, parent_weight: float) -> None
```

Sets DeMorgan parent weight

```
set_de_morgan_prior_belief(self: pysmile.Network, node_handle: int, belief: float) -> None
```

```
set_de_morgan_prior_belief(self: pysmile.Network, node_id: str, belief: float) -> None
```

Sets DeMorgan prior belief

```
set_default_outcome(self: pysmile.Network, node_handle: int, def_outcome: int) -> None
```

```
set_default_outcome(self: pysmile.Network, node_id: str, def_outcome: int) -> None
```

```
set_default_outcome(self: pysmile.Network, node_handle: int, def_outcome_id: str) -> None
```

```
set_default_outcome(self: pysmile.Network, node_id: str, def_outcome_id: str) -> None
```

Sets default outcome

```
set_description(self: pysmile.Network, new_description: str) -> None
```

Sets network description

```
set_discretization_sample_count(self: pysmile.Network, discretization_sample_count: int) -> None
```

Sets sample count for discretization

```
set_epis_params(self: pysmile.Network, epis_params: pysmile.EPISParams) -> None
```

Sets EPIS sampling parameters

```
set_evidence(self: pysmile.Network, node_handle: int, evidence_number: int) -> None
```

```
set_evidence(self: pysmile.Network, node_handle: int, evidence_id: str) -> None
```

```
set_evidence(self: pysmile.Network, node_id: str, evidence_number: int) -> None
```

```
set_evidence(self: pysmile.Network, node_id: str, evidence_id: str) -> None
```

Sets evidence on a node

```
set_ext_functions(self: pysmile.Network, external_functions: List[str]) -> None
```

Sets external functions

```
set_fault_outcome(self: pysmile.Network, node_handle: int, outcome_index: int, fault: bool) -> None
```

```
set_fault_outcome(self: pysmile.Network, node_id: str, outcome_index: int, fault: bool) -> None
```

```
set_fault_outcome(self: pysmile.Network, node_handle: int, outcome_id: str, fault: bool) -> None
```

```
set_fault_outcome(self: pysmile.Network, node_id: str, outcome_id: str, fault: bool) -> None
```

Sets fault outcome

```
set_id(self: pysmile.Network, new_id: str) -> None
```

Sets network ID

```
set_influence_diagram_algorithm(self: pysmile.Network, influence_diagram_algorithm:
```

```
pysmile.InfluenceDiagramAlgorithmType) -> None
```

Sets influence diagram algorithm

```
set_intervals(self: pysmile.Network, node_handle: int, intervals: List[float], remove_ids: bool) -> None
set_intervals(self: pysmile.Network, node_id: str, intervals: List[float], remove_ids: bool) -> None
```

Sets interval values

```
set_mandatory(self: pysmile.Network, node_handle: int, mandatory: bool) -> None
set_mandatory(self: pysmile.Network, node_id: str, mandatory: bool) -> None
```

Sets node as mandatory

```
set_mau_expressions(self: pysmile.Network, node_handle: int, expressions: List[str]) -> None
set_mau_expressions(self: pysmile.Network, node_id: str, expressions: List[str]) -> None
```

Sets MAU expressions

```
set_name(self: pysmile.Network, new_name: str) -> None
```

Sets network name

```
set_node_bg_color(self: pysmile.Network, node_handle: int, color: int) -> None
set_node_bg_color(self: pysmile.Network, node_id: str, color: int) -> None
```

Sets node background color

```
set_node_border_color(self: pysmile.Network, node_handle: int, color: int) -> None
set_node_border_color(self: pysmile.Network, node_id: str, color: int) -> None
```

Sets node border color

```
set_node_border_width(self: pysmile.Network, node_handle: int, width: int) -> None
set_node_border_width(self: pysmile.Network, node_id: str, width: int) -> None
```

Sets node border width

```
set_node_chart_enabled(self: pysmile.Network, node_handle: int, enabled: bool) -> None
set_node_chart_enabled(self: pysmile.Network, node_id: str, enabled: bool) -> None
```

Enables or disables node chart

```
set_node_chart_size(self: pysmile.Network, node_handle: int, width: int, height: int) -> None
set_node_chart_size(self: pysmile.Network, node_id: str, width: int, height: int) -> None
```

Sets node chart size

```
set_node_cost(self: pysmile.Network, node_handle: int, cost: List[float]) -> None
set_node_cost(self: pysmile.Network, node_id: str, cost: List[float]) -> None
```

Sets node cost

```
set_node_definition(self: pysmile.Network, node_handle: int, node_definition: List[float]) -> None
set_node_definition(self: pysmile.Network, node_id: str, node_definition: List[float]) -> None
```

Sets node definition table

```
set_node_description(self: pysmile.Network, node_handle: int, new_node_description: str) -> None
set_node_description(self: pysmile.Network, node_id: str, new_node_description: str) -> None
```

Sets node description

```
set_node_diag_type(self: pysmile.Network, node_handle: int, type: pysmile.NodeDiagType) -> None
set_node_diag_type(self: pysmile.Network, node_id: str, type: pysmile.NodeDiagType) -> None
```

Sets node diagnostic type

```
set_node_documentation(self: pysmile.Network, node_handle: int, documentation: List[pysmile.DocItemInfo]) -> None
set_node_documentation(self: pysmile.Network, node_id: str, documentation: List[pysmile.DocItemInfo]) -> None
```

Sets node documentation

```
set_node_equation(self: pysmile.Network, node_handle: int, equation: str) -> None
set_node_equation(self: pysmile.Network, node_id: str, equation: str) -> None
```

Sets node equation

```
set_node_equation_bounds(self: pysmile.Network, node_handle: int, lo: float, hi: float) -> None
set_node_equation_bounds(self: pysmile.Network, node_id: str, lo: float, hi: float) -> None
```

Sets bounds for node equation

```
set_node_equation_discretization(self: pysmile.Network, node_handle: int, intervals:
List[pysmile.DiscretizationInterval]) -> None
set_node_equation_discretization(self: pysmile.Network, node_id: str, intervals:
List[pysmile.DiscretizationInterval]) -> None
```

Sets discretization method

```
set_node_equation_uniform_discretization(self: pysmile.Network, node_handle: int, interval_count: int) -> None
set_node_equation_uniform_discretization(self: pysmile.Network, node_id: str, interval_count: int) -> None
```

Discretizes a continuous equation node using uniform intervals

```
set_node_id(self: pysmile.Network, node_handle: int, new_id: str) -> None
set_node_id(self: pysmile.Network, old_id: str, new_id: str) -> None
```

Sets ID of a node

```
set_node_name(self: pysmile.Network, arg0: int, arg1: str) -> None
set_node_name(self: pysmile.Network, arg0: str, arg1: str) -> None
```

Sets name of a node

```
set_node_position(self: pysmile.Network, node_handle: int, x: int, y: int, width: int, height: int) -> None
set_node_position(self: pysmile.Network, node_id: str, x: int, y: int, width: int, height: int) -> None
set_node_position(self: pysmile.Network, node_handle: int, rect: List[int]) -> None
set_node_position(self: pysmile.Network, node_id: str, rect: List[int]) -> None
```

Sets position of a node

```
set_node_question(self: pysmile.Network, node_handle: int, question: str) -> None
set_node_question(self: pysmile.Network, node_id: str, question: str) -> None
```

Sets question string

```
set_node_temporal_definition(self: pysmile.Network, node_handle: int, order: int, definition: List[float]) ->
None
set_node_temporal_definition(self: pysmile.Network, node_id: str, order: int, definition: List[float]) -> None
```

Sets temporal definition table

```
set_node_temporal_type(self: pysmile.Network, node_handle: int, temporal_type: pysmile.NodeTemporalType) ->
None
set_node_temporal_type(self: pysmile.Network, node_id: str, temporal_type: pysmile.NodeTemporalType) -> None
```

Sets temporal type

```
set_node_text_color(self: pysmile.Network, node_handle: int, color: int) -> None
set_node_text_color(self: pysmile.Network, node_id: str, color: int) -> None
```

Sets node text color

```
set_node_type(self: pysmile.Network, node_handle: int, new_node_type: int) -> None
set_node_type(self: pysmile.Network, node_id: str, new_node_type: int) -> None
```

Sets type of node

```
set_node_user_properties(self: pysmile.Network, node_handle: int, properties: List[pysmile.UserProperty]) -> None
```

```
set_node_user_properties(self: pysmile.Network, node_id: str, properties: List[pysmile.UserProperty]) -> None
```

Sets user properties of node

```
set_noisy_decomposition_enabled(self: pysmile.Network, enabled: bool) -> None
```

Enables noisy decomposition

```
set_noisy_decomposition_limit(self: pysmile.Network, limit: int) -> None
```

Sets limit on noisy decomposition

```
set_noisy_parent_strengths(self: pysmile.Network, node_handle: int, parent_index: int, strengths: List[int]) -> None
set_noisy_parent_strengths(self: pysmile.Network, node_id: str, parent_index: int, strengths: List[int]) -> None
set_noisy_parent_strengths(self: pysmile.Network, node_handle: int, parent_id: str, strengths: List[int]) -> None
set_noisy_parent_strengths(self: pysmile.Network, node_id: str, parent_id: str, strengths: List[int]) -> None
```

Sets noisy parent strengths

```
set_noisy_synchronized(self: pysmile.Network, node_handle: int, enable: bool) -> None
set_noisy_synchronized(self: pysmile.Network, node_id: str, enable: bool) -> None
```

Synchronizes noisy strengths

```
set_noisy_temporal_parent_strengths(self: pysmile.Network, node_handle: int, order: int, strengths: List[List[int]]) -> None
set_noisy_temporal_parent_strengths(self: pysmile.Network, node_id: str, order: int, strengths: List[List[int]]) -> None
```

Sets noisy temporal strengths

```
set_outcome_description(self: pysmile.Network, node_handle: int, outcome_index: int, description: str) -> None
set_outcome_description(self: pysmile.Network, node_id: str, outcome_index: int, description: str) -> None
set_outcome_description(self: pysmile.Network, node_handle: int, outcome_id: str, description: str) -> None
set_outcome_description(self: pysmile.Network, node_id: str, outcome_id: str, description: str) -> None
```

Sets outcome description

```
set_outcome_documentation(self: pysmile.Network, node_handle: int, outcome_index: int, documentation: List[pysmile.DocItemInfo]) -> None
set_outcome_documentation(self: pysmile.Network, node_handle: int, outcome_id: str, documentation: List[pysmile.DocItemInfo]) -> None
set_outcome_documentation(self: pysmile.Network, node_id: str, outcome_id: str, documentation: List[pysmile.DocItemInfo]) -> None
set_outcome_documentation(self: pysmile.Network, node_id: str, outcome_index: int, documentation: List[pysmile.DocItemInfo]) -> None
```

Sets outcome documentation

```
set_outcome_fix(self: pysmile.Network, node_handle: int, outcome_index: int, treatment: str) -> None
set_outcome_fix(self: pysmile.Network, node_id: str, outcome_index: int, treatment: str) -> None
set_outcome_fix(self: pysmile.Network, node_handle: int, outcome_id: str, treatment: str) -> None
set_outcome_fix(self: pysmile.Network, node_id: str, outcome_id: str, treatment: str) -> None
```

Fixes an outcome value

```
set_outcome_id(self: pysmile.Network, node_handle: int, old_outcome_id: str, new_outcome_id: str) -> None
set_outcome_id(self: pysmile.Network, node_id: str, old_outcome_id: str, new_outcome_id: str) -> None
set_outcome_id(self: pysmile.Network, node_handle: int, outcome_index: int, new_outcome_id: str) -> None
set_outcome_id(self: pysmile.Network, node_id: str, outcome_index: int, new_outcome_id: str) -> None
```

Sets ID of an outcome

```
set_outcome_label(self: pysmile.Network, node_handle: int, outcome_index: int, label: str) -> None
set_outcome_label(self: pysmile.Network, node_id: str, outcome_index: int, label: str) -> None
set_outcome_label(self: pysmile.Network, node_handle: int, outcome_id: str, label: str) -> None
set_outcome_label(self: pysmile.Network, node_id: str, outcome_id: str, label: str) -> None
```

Sets outcome label

```
set_outlier_rejection_enabled(self: pysmile.Network, enable: bool) -> None
```

Enables outlier rejection

```
set_point_values(self: pysmile.Network, node_handle: int, point_values: List[float]) -> None
set_point_values(self: pysmile.Network, node_id: str, point_values: List[float]) -> None
```

Sets point values for a node

```
set_rand_seed(self: pysmile.Network, arg0: int) -> None
```

Sets the random seed

```
set_ranked(self: pysmile.Network, node_handle: int, ranked: bool) -> None
set_ranked(self: pysmile.Network, node_id: str, ranked: bool) -> None
```

Sets ranking mode

```
set_relevance_enabled(self: pysmile.Network, enable: bool) -> None
```

Enables relevance propagation

```
set_sample_count(self: pysmile.Network, sample_count: int) -> None
```

Sets sample count

```
set_slice_count(self: pysmile.Network, slice_count: int) -> None
```

Sets number of time slices

```
set_submodel_description(self: pysmile.Network, handle: int, description: str) -> None
set_submodel_description(self: pysmile.Network, id: str, description: str) -> None
```

Sets submodel description

```
set_submodel_id(self: pysmile.Network, handle: int, id: str) -> None
```

Sets submodel ID

```
set_submodel_name(self: pysmile.Network, handle: int, name: str) -> None
set_submodel_name(self: pysmile.Network, id: str, name: str) -> None
```

Sets submodel name

```
set_submodel_of_node(self: pysmile.Network, submodel_handle: int, node_handle: int) -> None
set_submodel_of_node(self: pysmile.Network, submodel_id: str, node_id: str) -> None
```

Assigns node to submodel

```
set_submodel_of_submodel(self: pysmile.Network, parent_handle: int, child_handle: int) -> None
set_submodel_of_submodel(self: pysmile.Network, parent_id: str, child_id: str) -> None
```

Assigns submodel to parent submodel

```
set_submodel_position(self: pysmile.Network, submodel_handle: int, x: int, y: int, width: int, height: int) -> None
set_submodel_position(self: pysmile.Network, submodel_id: str, x: int, y: int, width: int, height: int) -> None
set_submodel_position(self: pysmile.Network, submodel_handle: int, rect: List[int]) -> None
set_submodel_position(self: pysmile.Network, submodel_id: str, rect: List[int]) -> None
```

Sets submodel position

```
set_target(self: pysmile.Network, node_handle: int, is_target: bool) -> None
set_target(self: pysmile.Network, node_id: str, is_target: bool) -> None
```

Marks node as target

```
set_temporal_cont_evidence(self: pysmile.Network, node_handle: int, slice: int, evidence: float) -> None
set_temporal_cont_evidence(self: pysmile.Network, node_id: str, slice: int, evidence: float) -> None
```

Sets temporal continuous evidence

```
set_temporal_evidence(self: pysmile.Network, node_handle: int, slice: int, outcome_index: int) -> None
set_temporal_evidence(self: pysmile.Network, node_id: str, slice: int, outcome_index: int) -> None
set_temporal_evidence(self: pysmile.Network, node_handle: int, slice: int, outcome_id: str) -> None
set_temporal_evidence(self: pysmile.Network, node_id: str, slice: int, outcome_id: str) -> None
```

Sets temporal evidence

```
set_temporal_virtual_evidence(self: pysmile.Network, node_handle: int, slice: int, evidence_definition:
List[float]) -> None
set_temporal_virtual_evidence(self: pysmile.Network, node_id: str, slice: int, evidence_definition:
List[float]) -> None
```

Sets temporal virtual evidence

```
set_user_properties(self: pysmile.Network, properties: List[pysmile.UserProperty]) -> None
```

Sets user-defined network properties

```
set_virtual_evidence(self: pysmile.Network, node_handle: int, evidence: List[float]) -> None
set_virtual_evidence(self: pysmile.Network, node_id: str, evidence: List[float]) -> None
```

Sets virtual evidence on a node

```
set_zero_avoidance_enabled(self: pysmile.Network, enable: bool) -> None
```

Enables zero-avoidance

```
temporal_arc_exists(self: pysmile.Network, parent_handle: int, child_handle: int, order: int) -> bool
temporal_arc_exists(self: pysmile.Network, parent_id: str, child_id: str, order: int) -> bool
```

Checks if temporal arc exists

```
unroll(self: pysmile.Network) -> pysmile.UnrollResults
```

Unrolls the temporal network

```
update_beliefs(self: pysmile.Network) -> None
```

Updates all node beliefs

```
write_file(self: pysmile.Network, xdsl_file: str) -> None
```

Writes network to a file

```
write_string(self: pysmile.Network) -> str
```

Serializes network to a string

9.18 pysmile.NodeSensitivity

coeff_a

Sensitivity coefficient A

coeff_b

Sensitivity coefficient B

coeff_c

Sensitivity coefficient C

coeff_d

Sensitivity coefficient D

sensitivity

Overall node sensitivity

9.19 **pysmile.NodeTemporalType**

CONTEMPORAL

Contemporal node

ANCHOR

Anchor node

TERMINAL

Terminal node

PLATE

Plate node

9.20 **pysmile.NodeType**

CPT

Conditional probability table node

TRUTH_TABLE

Truth table node

DECISION

Decision node

UTILITY

Utility node

MAU

Multi-attribute utility node

NOISY_MAX

Noisy-MAX node

EQUATION

Equation-based node

DEMORGAN

DeMorgan logic node

9.21 `pysmile.ObservationInfo`

cost

Observation cost

entropy

Observation measure

fault_posteriors

Posterior probabilities of faults

info_gain

Information gain from observation

measure

Observation measure

node

Node associated with observation

observation_priors

Prior probabilities for observation

9.22 `pysmile.SMILEException`

`__cause__`

exception cause

`__context__`

exception context

`__suppress_context__`

`__traceback__`

`__weakref__`

list of weak references to the object (if defined)

args

9.23 `pysmile.SensitivityResults`

`get_node_sensitivity(self: pysmile.SensitivityResults, src_handle: int, dst_handle: int, dst_outcome_index: int) -> pysmile.NodeSensitivity`

`get_node_sensitivity(self: pysmile.SensitivityResults, src_handle: int, dst_id: str, dst_outcome_id: str) ->`

```
pysmile.NodeSensitivity
get_node_sensitivity(self: pysmile.SensitivityResults, src_id: str, dst_id: str, dst_outcome_id: str) ->
pysmile.NodeSensitivity
get_node_sensitivity(self: pysmile.SensitivityResults, src_id: str, dst_handle: int, dst_outcome_index: int) ->
pysmile.NodeSensitivity
Gets sensitivity data for a node
```

9.24 pysmile.SingleFaultAlgorithmType

NORM_CROSSENTROPY
Normalized cross-entropy

CROSSENTROPY
Cross-entropy

PROB_CHANGE
Probability change

9.25 pysmile.TemporalInfo

handle
Temporal handle

id
Temporal ID

order
Temporal order

9.26 pysmile.UnrollResults

mapping
Mapping of unrolled nodes

unrolled
List of unrolled nodes

9.27 pysmile.UserProperty

name
Property name

value
Property value

9.28 pysmile.ValueOfInfo

```
add_node(self: pysmile.ValueOfInfo, node_handle: int) -> None
add_node(self: pysmile.ValueOfInfo, node_id: str) -> None
```

Adds a node to value of information analysis

`get_all_action_ids(self: pysmile.ValueOfInfo) -> List[str]`

Gets IDs of all actions

`get_all_actions(self: pysmile.ValueOfInfo) -> List[int]`

Gets all action nodes

`get_all_decision_ids(self: pysmile.ValueOfInfo) -> List[str]`

Gets IDs of all decision nodes

`get_all_decisions(self: pysmile.ValueOfInfo) -> List[int]`

Gets all decision nodes

`get_all_node_ids(self: pysmile.ValueOfInfo) -> List[str]`

Gets IDs of all nodes in VOI

`get_all_nodes(self: pysmile.ValueOfInfo) -> List[int]`

Gets all nodes in VOI

`get_decision(self: pysmile.ValueOfInfo) -> int`

Gets a specific decision node

`get_decision_id(self: pysmile.ValueOfInfo) -> str`

Gets the ID of a decision node

`get_indexing_node_ids(self: pysmile.ValueOfInfo) -> List[str]`

Gets IDs of indexing nodes

`get_indexing_nodes(self: pysmile.ValueOfInfo) -> List[int]`

Gets indexing nodes

`get_point_of_view(self: pysmile.ValueOfInfo) -> int`

Gets current point of view node

`get_point_of_view_id(self: pysmile.ValueOfInfo) -> str`

Gets ID of current point of view

`get_values(self: pysmile.ValueOfInfo) -> List[float]`

Gets computed VOI values

`remove_node(self: pysmile.ValueOfInfo, node_handle: int) -> None`

`remove_node(self: pysmile.ValueOfInfo, node_id: str) -> None`

Removes a node from VOI

`set_decision(self: pysmile.ValueOfInfo, node_handle: int) -> None`

`set_decision(self: pysmile.ValueOfInfo, node_id: str) -> None`

Sets a decision node

`set_point_of_view(self: pysmile.ValueOfInfo, node_handle: int) -> None`

`set_point_of_view(self: pysmile.ValueOfInfo, node_id: str) -> None`

Sets point of view node

`update(self: pysmile.ValueOfInfo) -> None`

Updates VOI analysis

9.29 Learning

TODO: provide some general description of the `pysmile.learning` package.

9.29.1 `pysmile.learning.BayesianSearch`

`get_bk_knowledge(self: pysmile.learning.BayesianSearch) -> pysmile.learning.BkKnowledge`

Gets background knowledge object

`get_iteration_count(self: pysmile.learning.BayesianSearch) -> int`

Gets the number of search iterations

`get_last_score(self: pysmile.learning.BayesianSearch) -> float`

Gets the score of the last learned structure

`get_link_probability(self: pysmile.learning.BayesianSearch) -> float`

Gets the link probability prior

`get_max_parents(self: pysmile.learning.BayesianSearch) -> int`

Gets the maximum number of parents for any node

`get_max_search_time(self: pysmile.learning.BayesianSearch) -> int`

Gets the maximum search time

`get_prior_link_probability(self: pysmile.learning.BayesianSearch) -> float`

Gets prior probability for link existence

`get_prior_sample_size(self: pysmile.learning.BayesianSearch) -> int`

Gets prior sample size

`get_rand_seed(self: pysmile.learning.BayesianSearch) -> int`

Gets the random seed used in search

`learn(self: pysmile.learning.BayesianSearch, dataset: pysmile.learning.DataSet) -> pysmile.Network`

Performs Bayesian structure learning

`set_bk_knowledge(self: pysmile.learning.BayesianSearch, bk_knowledge: pysmile.learning.BkKnowledge) -> None`

Sets background knowledge object

`set_iteration_count(self: pysmile.learning.BayesianSearch, iteration_count: int) -> None`

Sets the number of search iterations

`set_link_probability(self: pysmile.learning.BayesianSearch, link_probability: float) -> None`

Sets the link probability prior

`set_max_parents(self: pysmile.learning.BayesianSearch, max_parents: int) -> None`

Sets the maximum number of parents for any node

`set_max_search_time(self: pysmile.learning.BayesianSearch, max_search_time: int) -> None`

Sets the maximum search time

`set_prior_link_probability(self: pysmile.learning.BayesianSearch, link_probability: float) -> None`

Sets prior probability for link existence

`set_prior_sample_size(self: pysmile.learning.BayesianSearch, prior_sample_size: int) -> None`

Sets prior sample size

`set_rand_seed(self: pysmile.learning.BayesianSearch, rand_seed: int) -> None`

Sets the random seed used in search

9.29.2 pysmile.learning.BkKnowledge

`add_forbidden_arc(self: pysmile.learning.BkKnowledge, from: int, to: int) -> None`

`add_forbidden_arc(self: pysmile.learning.BkKnowledge, from: str, to: str) -> None`

Adds a forbidden arc between two variables

`add_forced_arc(self: pysmile.learning.BkKnowledge, from: int, to: int) -> None`

`add_forced_arc(self: pysmile.learning.BkKnowledge, from: str, to: str) -> None`

Adds a forced arc between two variables

`add_variable(self: pysmile.learning.BkKnowledge, id: str) -> None`

Adds a variable to background knowledge

`find_variable(self: pysmile.learning.BkKnowledge, id: str) -> int`

Finds a variable by name

`get_forbidden_children(self: pysmile.learning.BkKnowledge, var: int) -> List[int]`

Gets forbidden children of a variable

`get_forced_children(self: pysmile.learning.BkKnowledge, var: int) -> List[int]`

Gets forced children of a variable

`get_tier(self: pysmile.learning.BkKnowledge, var: int) -> int`

`get_tier(self: pysmile.learning.BkKnowledge, var: str) -> int`

Gets the tier of a variable

`get_variable_count(self: pysmile.learning.BkKnowledge) -> int`

Gets the number of variables

`get_variable_id(self: pysmile.learning.BkKnowledge, variable: int) -> str`

Gets the ID of a variable

`match_data(self: pysmile.learning.BkKnowledge, dataset: pysmile.learning.DataSet) -> None`

Matches the knowledge with a dataset

`read_file(self: pysmile.learning.BkKnowledge, filename: str) -> None`

Reads background knowledge from a file

`set_tier(self: pysmile.learning.BkKnowledge, var: int, tier: int) -> None`

`set_tier(self: pysmile.learning.BkKnowledge, var: str, tier: int) -> None`

Sets the tier of a variable

```
write_file(self: pysmile.learning.BkKnowledge, filename: str) -> None
```

Writes background knowledge to a file

9.29.3 pysmile.learning.Curve

stat

Summary statistic of the curve (area under the curve)

x

X-coordinates of the curve

y

Y-coordinates of the curve

9.29.4 pysmile.learning.DataMatch

column

Gets column index in data

node

Gets matched node ID

slice

Gets data slice index

9.29.5 pysmile.learning.DataSet

```
add_empty_record(self: pysmile.learning.DataSet) -> None
```

Adds an empty record to the dataset

```
add_float_variable(self: pysmile.learning.DataSet, id: str, missing_value: float) -> None
```

```
add_float_variable(self: pysmile.learning.DataSet, id: str) -> None
```

Adds a continuous variable

```
add_int_variable(self: pysmile.learning.DataSet, id: str, missing_value: int) -> None
```

```
add_int_variable(self: pysmile.learning.DataSet, id: str) -> None
```

Adds a discrete variable

```
discretize(self: pysmile.learning.DataSet, variable: int, algorithm:
pysmile.learning.DataSet.DiscretizationAlgorithmType, intervals: int, state_prefix: str) -> List[float]
```

Discretizes continuous variables

```
find_variable(self: pysmile.learning.DataSet, variable_id: str) -> int
```

Finds variable by name

```
get_float(self: pysmile.learning.DataSet, variable: int, record: int) -> float
```

Gets float value from dataset

get_int(self: pysmile.learning.DataSet, variable: int, record: int) -> int

Gets integer value from dataset

get_record_count(self: pysmile.learning.DataSet) -> int

Gets number of records

get_state_names(self: pysmile.learning.DataSet, variable: int) -> List[str]

Gets state names of a discrete variable

get_variable_count(self: pysmile.learning.DataSet) -> int

Gets number of variables

get_variable_id(self: pysmile.learning.DataSet, variable_index: int) -> str

Gets variable ID

is_discrete(self: pysmile.learning.DataSet, variable_index: int) -> bool

is_discrete(self: pysmile.learning.DataSet, variable_id: str) -> bool

Checks if a variable is discrete

is_missing(self: pysmile.learning.DataSet, variable: int, record: int) -> bool

Checks if a value is missing

match_network(self: pysmile.learning.DataSet, net: pysmile.Network) -> List[pysmile.learning.DataMatch]

Matches dataset to a network structure

read_file(self: pysmile.learning.DataSet, filename: str, missing_value_token: str, missing_int: int, missing_float: float, column_ids_present: bool) -> None

read_file(self: pysmile.learning.DataSet, filename: str, missing_value_token: str) -> None

read_file(self: pysmile.learning.DataSet, file_name: str) -> None

Reads dataset from file

read_pandas_dataframe(self: pysmile.learning.DataSet, *args, **kwargs) -> None

Reads data from a pandas DataFrame

set_float(self: pysmile.learning.DataSet, variable: int, record: int, value: float) -> None

Sets float value in dataset

set_int(self: pysmile.learning.DataSet, variable: int, record: int, value: int) -> None

Sets integer value in dataset

set_missing(self: pysmile.learning.DataSet, variable: int, record: int) -> None

Marks a value as missing

set_state_names(self: pysmile.learning.DataSet, variable: int, names: List[str]) -> None

Sets state names for a variable

write_file(self: pysmile.learning.DataSet, filename: str, separator: str, missing_value_token: str, column_ids_present: bool) -> None

write_file(self: pysmile.learning.DataSet, filename: str) -> None

Writes dataset to file

9.29.6 pysmile.learning.EM

`get_auto_slices(self: pysmile.learning.EM) -> bool`

Gets whether auto slicing is enabled

`get_eq_sample_size(self: pysmile.learning.EM) -> int`

Gets equivalent sample size for EM

`get_last_score(self: pysmile.learning.EM) -> float`

Gets score after last EM run

`get_randomize_parameters(self: pysmile.learning.EM) -> bool`

Checks if EM should randomize parameters

`get_relevance(self: pysmile.learning.EM) -> bool`

Gets relevance factor for EM

`get_seed(self: pysmile.learning.EM) -> int`

Gets random seed for EM

`get_uniformize_parameters(self: pysmile.learning.EM) -> bool`

Checks if parameters should be uniformized before EM

`learn(self: pysmile.learning.EM, data: pysmile.learning.DataSet, net: pysmile.Network, matching: List[pysmile.learning.DataMatch], fixed_nodes: List[int]) -> None`
`learn(self: pysmile.learning.EM, data: pysmile.learning.DataSet, net: pysmile.Network, matching: List[pysmile.learning.DataMatch], fixed_nodes: List[str]) -> None`
`learn(self: pysmile.learning.EM, data: pysmile.learning.DataSet, net: pysmile.Network, matching: List[pysmile.learning.DataMatch]) -> None`

Performs EM learning

`set_auto_slices(self: pysmile.learning.EM, value: bool) -> None`

Enables or disables auto slicing

`set_eq_sample_size(self: pysmile.learning.EM, size: int) -> None`

Sets equivalent sample size for EM

`set_randomize_parameters(self: pysmile.learning.EM, value: bool) -> None`

Enables or disables random parameter initialization

`set_relevance(self: pysmile.learning.EM, value: bool) -> None`

Sets relevance factor for EM

`set_seed(self: pysmile.learning.EM, seed: int) -> None`

Sets random seed for EM

`set_uniformize_parameters(self: pysmile.learning.EM, value: bool) -> None`

Enables or disables parameter uniformization

9.29.7 pysmile.learning.GreedyThickThinning

`get_bk_knowledge(self: pysmile.learning.GreedyThickThinning) -> pysmile.learning.BkKnowledge`

Gets background knowledge object

`get_max_parents(self: pysmile.learning.GreedyThickThinning) -> int`

Gets maximum number of parents per node

`learn(self: pysmile.learning.GreedyThickThinning, data: pysmile.learning.DataSet) -> pysmile.Network`

Performs greedy thick thinning structure learning

`set_bk_knowledge(self: pysmile.learning.GreedyThickThinning, bkk: pysmile.learning.BkKnowledge) -> None`

Sets background knowledge object

`set_max_parents(self: pysmile.learning.GreedyThickThinning, count: int) -> None`

Sets maximum number of parents per node

9.29.8 pysmile.learning.NaiveBayes

`get_class_variable_id(self: pysmile.learning.NaiveBayes) -> str`

Gets class variable ID

`learn(self: pysmile.learning.NaiveBayes, data: pysmile.learning.DataSet) -> pysmile.Network`

Learns a Naive Bayes model

`set_class_variable_id(self: pysmile.learning.NaiveBayes, id: str) -> None`

Sets class variable ID

9.29.9 pysmile.learning.PC

`get_bk_knowledge(self: pysmile.learning.PC) -> pysmile.learning.BkKnowledge`

Gets background knowledge object

`get_max_adjacency(self: pysmile.learning.PC) -> int`

Gets maximum adjacency constraint

`get_max_search_time(self: pysmile.learning.PC) -> int`

Gets the maximum search time

`get_significance(self: pysmile.learning.PC) -> float`

Gets significance level for conditional independence tests

`learn(self: pysmile.learning.PC, data: pysmile.learning.DataSet) -> pysmile.learning.Pattern`

Performs PC structure learning

`set_bk_knowledge(self: pysmile.learning.PC, bkk: pysmile.learning.BkKnowledge) -> None`

Sets background knowledge object

`set_max_adjacency(self: pysmile.learning.PC, adjacency: int) -> None`

Sets maximum adjacency constraint

`set_max_search_time(self: pysmile.learning.PC, search_time: int) -> None`

Sets the maximum search time

`set_significance(self: pysmile.learning.PC, sig: float) -> None`

Sets significance level for conditional independence tests

9.29.10 `pysmile.learning.Pattern`

`get_edge(self: pysmile.learning.Pattern, from: int, to: int) -> pysmile.learning.Pattern.EdgeType`

Gets the edge type between two nodes

`get_size(self: pysmile.learning.Pattern) -> int`

Gets the number of nodes in the pattern

`has_cycle(self: pysmile.learning.Pattern) -> bool`

Checks if the pattern contains a cycle

`is_dag(self: pysmile.learning.Pattern) -> bool`

Checks if the pattern is a DAG

`make_network(self: pysmile.learning.Pattern, ds: pysmile.learning.DataSet) -> pysmile.Network`

Creates a network from the pattern

`set_edge(self: pysmile.learning.Pattern, from: int, to: int, type: pysmile.learning.Pattern.EdgeType) -> None`

Sets the edge type between two nodes

`set_size(self: pysmile.learning.Pattern, size: int) -> None`

Sets the number of nodes in the pattern

9.29.11 `pysmile.learning.TAN`

`get_class_variable_id(self: pysmile.learning.TAN) -> str`

Gets class variable ID

`get_max_search_time(self: pysmile.learning.TAN) -> int`

Gets maximum time allowed for TAN search

`get_rand_seed(self: pysmile.learning.TAN) -> int`

Gets random seed for TAN structure learning

`learn(self: pysmile.learning.TAN, data: pysmile.learning.DataSet) -> pysmile.Network`

Learns a Tree Augmented Naive Bayes (TAN) model

`set_class_variable_id(self: pysmile.learning.TAN, id: str) -> None`

Sets class variable ID

`set_max_search_time(self: pysmile.learning.TAN, search_time: int) -> None`

Sets maximum time allowed for TAN search

```
set_rand_seed(self: pysmile.learning.TAN, seed: int) -> None
```

Sets random seed for TAN structure learning

9.29.12 pysmile.learning.Validator

```
add_class_node(self: pysmile.learning.Validator, node_handle: int) -> None
```

```
add_class_node(self: pysmile.learning.Validator, node_id: str) -> None
```

Adds class node to validator

```
get_accuracy(self: pysmile.learning.Validator, node_handle: int, outcome_index: int) -> float
```

```
get_accuracy(self: pysmile.learning.Validator, node_id: str, outcome_index: int) -> float
```

```
get_accuracy(self: pysmile.learning.Validator, node_handle: int, outcome_id: str) -> float
```

```
get_accuracy(self: pysmile.learning.Validator, node_id: str, outcome_id: str) -> float
```

Gets classification accuracy

```
get_calibration_curve_b(self: pysmile.learning.Validator, node_handle: int, outcome_index: int, bin_count: int)
-> pysmile.learning.Curve
```

```
get_calibration_curve_b(self: pysmile.learning.Validator, node_id: str, outcome_index: int, bin_count: int) ->
pysmile.learning.Curve
```

```
get_calibration_curve_b(self: pysmile.learning.Validator, node_handle: int, outcome_id: str, bin_count: int) ->
pysmile.learning.Curve
```

```
get_calibration_curve_b(self: pysmile.learning.Validator, node_id: str, outcome_id: str, bin_count: int) ->
pysmile.learning.Curve
```

Returns the calibration curve by binning for the specified node, outcome and bin count

```
get_calibration_curve_ma(self: pysmile.learning.Validator, node_handle: int, outcome_index: int, window_size:
int) -> pysmile.learning.Curve
```

```
get_calibration_curve_ma(self: pysmile.learning.Validator, node_id: str, outcome_index: int, window_size: int)
-> pysmile.learning.Curve
```

```
get_calibration_curve_ma(self: pysmile.learning.Validator, node_handle: int, outcome_id: str, window_size: int)
-> pysmile.learning.Curve
```

```
get_calibration_curve_ma(self: pysmile.learning.Validator, node_id: str, outcome_id: str, window_size: int) ->
pysmile.learning.Curve
```

Returns the calibration curve by moving average for the specified node, outcome and window size

```
get_confusion_matrix(self: pysmile.learning.Validator, node_handle: int) -> List[List[int]]
```

```
get_confusion_matrix(self: pysmile.learning.Validator, node_id: str) -> List[List[int]]
```

Gets confusion matrix

```
get_result_data_set(self: pysmile.learning.Validator) -> pysmile.learning.DataSet
```

Gets result dataset after validation

```
get_roc(self: pysmile.learning.Validator, node_handle: int, outcome_index: int) -> pysmile.learning.Curve
```

```
get_roc(self: pysmile.learning.Validator, node_id: str, outcome_index: int) -> pysmile.learning.Curve
```

```
get_roc(self: pysmile.learning.Validator, node_handle: int, outcome_id: str) -> pysmile.learning.Curve
```

```
get_roc(self: pysmile.learning.Validator, node_id: str, outcome_id: str) -> pysmile.learning.Curve
```

Returns the ROC curve for the specified node and outcome

```
k_fold(self: pysmile.learning.Validator, em: pysmile.learning.EM, fold_count: int, folding_rand_seed: int) ->
None
```

```
k_fold(self: pysmile.learning.Validator, em: pysmile.learning.EM, fold_count: int) -> None
```

Performs k-fold cross-validation

```
leave_one_out(self: pysmile.learning.Validator, em: pysmile.learning.EM) -> None
```

Performs leave-one-out cross-validation

```
test(self: pysmile.learning.Validator) -> None
```

Tests the model on a dataset

Acknowledgments

10 Acknowledgments

SMILE internally uses portions of the following two software libraries: micro-ECC and Expat. PySMILE uses pybind11. These three libraries require an acknowledgment that we are reproducing below.

micro-ECC

Copyright (c) 2014, Kenneth MacKay

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Expat

Copyright (c) 1998-2000 Thai Open Source Software Center Ltd and Clark Cooper

Copyright (c) 2001-2017 Expat maintainers

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

pybind

Copyright (c) 2016 Wenzel Jakob <wenzel.jakob@epfl.ch>, All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.