SMILE Wrappers

Programmer's Manual

Version 1.0.3, Built on 7/9/2018
BayesFusion, LLC

This page is intentionally left blank.

Introduction

1 Introduction

Welcome to SMILE Wrapper Programmer's Manual, Version 1.0.3, Built on 7/9/2018.

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

SMILE is a C++ software library for performing Bayesian inference. In order to ensure that its functionality can be easily integrated into software written in other languages, BayesFusion, LLC, provides wrapper libraries for Java, Python and .NET. The names of these products are:

- jSMILE (Java and environments which can instantiate and use the JVM, such as R)
- PySMILE (Python)
- SMILE.NET (.NET)

Each wrapper includes SMILE, so you do not need to download SMILE or know how to use C++ SMILE in order to use the wrappers. We strive to ensure feature symmetry between the wrappers - features available in one wrapper are generally available in other wrappers.

We have also developed SMILE.COM, a wrapper exposing SMILE functionality through Windows' COM (Common Object Model). The target audience for SMILE.COM are Microsoft Excel users, although the library will work with any environment extensible through COM. This manual does not contain documentation for SMILE.COM.

If you are new to SMILE and would like to begin with an informal, tutorial-like introduction, please start with the Java, Python or .NET section of *Platforms and Wrappers* 10 (depending on the programming language you're going to use), followed by *Hello SMILE Wrapper!* 14 section. If you are an advanced user, please browse through the *Table of Contents* or search for the topic of your interest.

This manual refers to a good number of concepts that are assumed to be known to the reader, such as probability, utility, decision theory and decision analysis, Bayesian networks, influence diagrams, etc. Should you want to learn more about these, please refer to GeNIe manual. SMILE is GeNIe's Application Programmer's Interface (API) and practically every elementary operation performed with GeNIe translates to calls to SMILE methods. Being familiar with GeNIe may prove extremely useful in learning SMILE. Understanding some of SMILE's functionality may be easier when performed interactively in GeNIe.

Licensing

2 Licensing

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

Deployment of SMILE library or SMILE wrappers, i.e., embedding them into user programs, requires a deployment license. There are two types of deployment licenses: Server license, which allows a program linked with SMILE to be deployed on a computer server, and end-user program license, which allows distribution of user programs that include SMILE. Please contact BayesFusion, LLC, for details of the licenses and pricing.

The licensing system is implemented as a fragment of code which contains your license key and must be executed as first call into SMILE wrapper that your program uses. **This code is not included in SMILE wrapper distribution**, it is personalized by BayesFusion, LLC, for you or your organization. The license keys are provided in a compressed .zip file, which contains Java, R, Python, C# and VB.NET code. 6-month academic and free 30-day evaluation licenses can be obtained directly at https://download.bayesfusion.com.

Platforms and Wrappers

3 Platforms and Wrappers

In addition to ensuring feature parity between wrappers, we designed them to use similiar names for the classes and methods. For example, all three wrappers contain the `Network` class. At the same time, wrappers honor the idioms of the specific languages they target:

- `Network` class in jSMILE has `addNode` method
- `Network` class in PySMILE has `add_node` method
- `Network` class in SMILE.NET has `AddNode` method

As you can see, the naming choices for the methods reflect the naming conventions of different programming languages.

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

3.1 Java and jSMILE

jSMILE is compatible with JDK 7 and newer.

jSMILE contains two parts: the jar file and the native library, the latter is 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 as any Java jar. You can add it to your local Maven repository, for example.

The native library is used by the code in the jar file through Java Native Interface (JNI). The name of the native library file 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 will attempt to load its native library. If the `jsmile.native.library` system property is set, jSMILE calls `System.load` method, which requires a complete path name of the native library (no relative paths are allowed), using the value of the property as the path name. Otherwise, the `System.loadLibrary` method is used. In such 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 set once JVM is running, but `jsmile.native.library` can be set 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 SMILE library

The second group of classes, which includes `smile.Network`, is derived from `smile.Wrapper`. Each object instance of these classes has a related C++ object, which is deallocated by the Java object's finalizer or by a call to `Wrapper.Dispose` method. To ensure early deallocation of C++ object resources associated with your Java object, `Dispose` should be called as soon as the program doesn't need the object anymore. If `Dispose` is not called, the deallocation will happen in the finalizer during garbage collection.

For details on using jSMILE from R, see the Appendix 70.

3.2 Python and PySMILE

PySMILE, which is compatible with Python 2, consists of a single dynamically loaded library. The library name is platform-dependent:

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

To use PySMILE, you just need to import the `pysmile` module:

```
import pysmile
```

Your license key can be pasted directly into your Python code, or imported:

```
import pysmile_license
```

We can provide binaries compatible with Python 3 on request.

3.3 .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 should be pasted into the sources of your program. We provide the keys as C# and VB.NET code.

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
}
```

Hello, SMILE Wrapper!

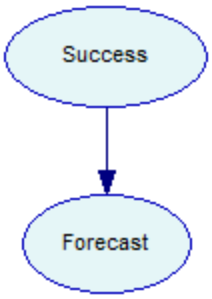
4 Hello, SMILE Wrapper!

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* [14] section below.

4.1 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.



Success	0.2
Failure	0.8

	Success	Failure
Good	0.4	0.1
Moderate	0.4	0.3
Poor	0.2	0.6

4.2 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>
```

4.3 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, C#) are included below. Note that you'll need to use your SMILE license key. See the *Licensing* [8] section of this manual if you want to obtain your academic or trial license key. See *Platforms and Wrappers* [10] 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:

```
Java:   net.readFile("VentureBN.xdsl");
Python: net.read_file("VentureBN.xdsl")
C#:     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:

```
Java:   net.setEvidence("Forecast", "Moderate");
Python: net.set_evidence("Forecast", "Moderate")
C#:     net.SetEvidence("Forecast", "Moderate");
```

Now we can update the network:

```
Java:   net.updateBeliefs();
Python: net.update_beliefs()
C#:     net.UpdateBeliefs();
```

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

```
Java:   double[] beliefs = net.getNodeValue("Success");
Python: beliefs = net.get_node_value("Success")
C#:     double[] 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 preceeded 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 Java are returned by the `Network.getOutcomeId` method. At this point you should easily derive the equivalent method names for Python and C# from Java 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* [32] section of this manual.

4.3.1 Hello.java

```
import smile.*;

public class Hello {
    public static void main(String[] args) {
        System.out.println("Paste your license 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]);
        }
    }
}
```

4.3.2 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()
```

4.3.3 Hello.cs

```
using System;
using Smile;

namespace SmileNetTutorial
{
    class Hello
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Paste your SMILE License 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]);
            }
        }
    }
}
```

This page is intentionally left blank.

Using SMILE Wrappers

5 Using SMILE Wrappers

5.1 Error handling

SMILE Wrappers throw exceptions when the underlying C++ SMILE calls fail. The exception types are:

```
Java:   smile.SMILEException
Python: pysmile.SMILEExcetpion
C#:     Smile.SmileException
```

5.2 Networks, nodes and arcs

The most important class defined by SMILE Wrappers is `Network`. The objects of this class act as containers for nodes and are responsible for node creation and destruction. Nodes and arcs are always created and destroyed by invoking the methods of the `Network` class. The access to existing nodes and arcs is always performed through the `Network` object where they live.

There are no classes representing nodes and arcs. The `Network` object works as a facade for the complex underlying data structure.

5.2.1 Network

To create a network, simply use a default constructor:

Java:

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

Python:

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

C#:

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

See the *Platforms and Wrappers* [10] section for the platform-specific information about the lifetime of instances of `Network` class.

5.2.2 Nodes

Use the following methods to add and delete nodes:

Java:

```
Network.addNode
Network.deleteNode
```

Python:

```
Network.add_node
Network.delete_node
```

C#:

```
Network.AddNode
Network.DeleteNode
```

When creating the node you need to specify its type.

Within the network, the nodes are uniquely identified by their handle. The node handle is a non-negative integer, which is preserved when network is copied. In addition to handles, each node has an unique (in the context of its containing network), persistent, textual identifier. This identifier is specified as an argument to `Network.addNode` method at node creation (it may be changed later). The identifiers in SMILE start are case-sensitive, start with a letter and contain letters, digits and underscores. Node's identifier can be converted to node handle with a call to `Network.getNode` method.

All methods of the `Network` class dealing with nodes can use either integer node handles or string identifiers to specify the nodes. However, the versions using identifiers will be internally performing O(N) string lookup (where N is the number of nodes in the network), while the versions using handlers are only using O(1) validity check.

The values of the handles are not guaranteed to be consecutive or start from any particular value. To iterate over nodes in the network, use `Network.getFirstNode` and `getNextNode`:

Java:

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

Python:

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

C#:

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

Note the loop exit condition - we proceed only if the returned handle is greater than or equal to zero.

Here are the methods to obtain an array of all node handles, an array of all node identifiers or a number of nodes in the network:

Java:

```
Network.getAllNodes
Network.getAllNodeIds
Network.getNodeCount
```

Python:

```
Network.get_all_nodes
Network.get_all_node_ids
Network.get_node_count
```

C#:

```
Network.GetAllNodes
Network.GetAllNodeIds
Network.NodeCount (read-only property)
```

Nodes may be marked as targets with `Network.setTarget` method. Target nodes are always guaranteed to be updated by the inference algorithm. Other nodes, i.e., nodes that are not designated as targets, may be updated or not, depending on the internals of the algorithm used, but are not guaranteed to be updated. Focusing inference on the target nodes can reduce time and memory required to complete the calculation. When no targets are specified, SMILE assumes that all nodes are of interest to the user.

5.2.3 Arcs

Here are the `Network` class methods to add and delete arcs between nodes:

Java:

```
Network.addArc
Network.deleteArc
```

Python:

```
Network.add_arc
Network.delete_arc
```

C#:

```
Network.AddArc
```

```
Network.DeleteArc
```

The graph defined by nodes and arcs in a `Network` class instance a directed acyclic graph (DAG) at all times. An attempt to add an arc which would create a cycle in the graph will cause an exception.

To inspect the graph structure, you can use the following methods:

Java:

```
Network.getParents
Network.getParentIds
Network.getChildren
Network.getChildIds
```

Python:

```
Network.get_parents
Network.get_parent_ids
Network.get_children
Network.get_child_ids
```

C#:

```
Network.GetParents
Network.GetParentIds
Network.GetChildren
Network.GetChildIds
```

The methods with the "Ids" suffix return the arrays of string identifiers of the related nodes, otherwise the arrays of integer node handles are used.

5.3 Anatomy of a node

The different aspects of a node available through `Network` class methods are:

- node definition
- node value
- node evidence
- attributes which do not affect inference, but determine node's location, color, name, etc.

The definition of the node specifies how it interacts with other nodes in the network. The node definition is written as part of the network when the network is saved to a file or serialized as a string variable. For general chance node the definition consists of conditional probability table (CPT) and list of state names.

The value of the node contains the values (typically, the marginal probability distribution or the expected utilities) calculated for the node by the inference algorithm. Unlike the definition, the value is not written as part of the network.

The node evidence

5.3.1 Node definition

The definition of the node specifies how it interacts with other nodes in the network. The node definition is written as part of the network when the network is saved to a file or serialized.

Use the following methods to read or write the numeric part of node definition:

Java:

```
Network.setNodeDefinition
Network.getNodeDefinition
```

Python:

```
Network.set_node_definition
Network.get_node_definition
```

C#:

```
Network.SetNodeDefinition
Network.GetNodeDefinition
```

The discrete nodes (including the general chance nodes) also have the list of outcomes as part of their definition. The outcomes are identified by string identifiers, which must be unique within the node and follow SMILE rules for identifier: are case-sensitive, start with a letter and contain letters, digits and underscores. Outcomes can be added, deleted or renamed.

5.3.2 Node value

The value of the node contains the values (typically, the marginal probability distribution or the expected utilities) calculated for the node by the inference algorithm. Unlike the definition, the value is not written as part of the network during I/O operations. In influence diagrams the node value is calculated for multiple sets of decisions. In such case, you also need to retrieve the indexing parents of node value to properly interpret the numbers. Note that value indexing parents are not the same as node parents. The value also stores its validity status; a flag set to true by the inference algorithm after it succesfully completes the calculations for the given node. You should not call any value-reading methods unless the value is valid.

Here are the methods related to node values:

Java:

```
Network.isValueValid
Network.getNodeValue
Network.getValueIndexingParents
Network.getValueIndexingParentIds
```

Python:

```
Network.is_value_valid
Network.get_node_value
Network.get_value_indexing_parents
Network.get_value_indexing_parent_ids
```

C#:

```
Network.IsValueValid
Network.getNodeValue
Network.getValueIndexingParents
Network.getValueIndexingParentIds
```

5.3.3 Node evidence

The output from the inference algorithms depends on the node definitions and the evidence set on nodes in the network. The evidence for discrete nodes is specified by outcome index or outcome identifier. For continuous nodes which support evidence, the evidence is a number.

Nodes can become propagated evidence (implied by other evidence set in the network).

Special type of evidence for discrete nodes is virtual evidence, which is a probability distribution over the outcomes of the node.

The methods related to evidence include:

Java:

```
Network.isEvidence
Network.isPropagatedEvidence
Network.isRealEvidence
Network.clearEvidence
Network.getEvidence
Network.getEvidenceId
Network.setEvidence
Network.getContEvidence
Network.setContEvidence
Network.isVirtualEvidence
Network.getVirtualEvidence
Network.setVirtualEvidence
```

Python:

```
Network.is_evidence
Network.is_propagated_evidence
```

```
Network.is_real_evidence
Network.clear_evidence
Network.get_evidence
Network.get_evidence_id
Network.set_evidence
Network.get_cont_evidence
Network.set_cont_evidence
Network.is_virtual_evidence
Network.get_virtual_evidence
Network.set_virtual_evidence
```

C#:

```
Network.IsEvidence
Network.IsPropagatedEvidence
Network.IsRealEvidence
Network.ClearEvidence
Network.GetEvidence
Network.GetEvidenceId
Network.SetEvidence
Network.GetContEvidence
Network.SetContEvidence
Network.IsVirtualEvidence
Network.GetVirtualEvidence
Network.SetVirtualEvidence
```

5.3.4 Other node attributes

(to be done)

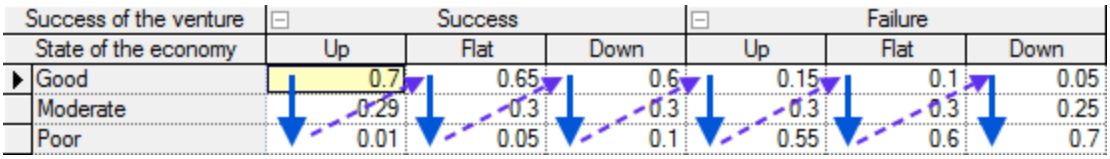
5.4 Multidimensional arrays

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 x 3 x 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 (or rather, the zero-th, because all indexes in SMILE are zero-based) element, the one with the value of 0.7 and yellow background, represents *P(Forecast=Good | Success of the venture=Success & State of the economy=Up)*. It is followed by the probabilties 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 the Network.getNodeDefinition is a single-dimensional array. If your program needs to convert between linear and multidimensional coordinates, see *Tutorial 3*[47]. 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.

5.5 Input and Output

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

The native format for SMILE networks is XDSL. The format is XML-based and the definition schema is available at BayesFusion documentation website (http://support.bayesfusion.com/docs/). When writing network in this format to file, the `.xdsl` extension should be used.

XDSL is the only format supported by string I/O methods. File-based methods can read and write other formats. Depending on the feature parity between SMILE and the 3rd party software using other file types, some of the information may be lost.

Java:

```
Network.readFile
Network.writeFile
Network.readString
Network.writeString
```

Python:

```
Network.read_file
Network.write_file
Network.read_string
Network.write_string
```

C#:

```
Network.ReadFile
Network.WriteFile
Network.ReadString
Network.WriteString
```

5.6 Inference

SMILE includes functions for several popular Bayesian network inference algorithms, including the clustering algorithm, and several approximate stochastic sampling algorithms. To run the inference or obtain the probability of evidence currently set in the network, use the following methods:

Java:

```
Network.updateBeliefs
Network.probEvidence
```

Python:

```
Network.update_beliefs
Network.prob_evidence
```

C#:

```
Network.UpdateBeliefs
Network.ProbEvidence
```

5.7 User Properties

(to be done)

5.8 Diagnosis

(to be done)

5.9 Dynamic Bayesian networks

(to be done)

5.10 Continuous models

(to be done)

5.11 Hybrid models

(to be done)

5.12 Learning

(to be done)

This page is intentionally left blank.

Tutorials

6 Tutorials

The complete, functionally equivalent source code for all the tutorials in Java, Python and C# is provided later in this chapter. The multi-line code snippets in the tutorial descriptions preceeding the source code are also given in Java, Python and C#. The inline examples in text paragraphs are using Java syntax (like `Network.updateBeliefs`) for brevity.

6.1 Tutorial 1: Creating a Bayesian Network

Consider a slight twist on the problem described in the *Hello, SMILE Wrapper!* [14] 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`.

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);
```

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)
```

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);
```

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

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;
```

```
}
```

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
```

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;
}
```

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.

Java:

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

Python:

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

C#:

```
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*[26] 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.

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);
```

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)
```

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);
```

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.

Java:

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

Python:

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

C#:

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

6.1.1 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;
    }
}
```

6.1.2 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):
        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=G|Success=F,Economy=D)
            0.70  # P(Forecast=G|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
```

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

            net.WriteFile("tutorial1.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;
        }
    }

}
```

6.2 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`.

Java:

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

Python:

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

C#:

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

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

Java:

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

Python:

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

C#:

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

Java:

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

Python:

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

C#:

```
for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
{
    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 is 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`).

6.2.1 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);
    }

}
```

6.2.2 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(node_id + " has evidence set (" +
                  net.get_outcome_id(node_handle,
                                     net.get_evidence(node_handle)) + ")")
        else:
            posteriors = net.get_node_value(node_handle)
            for i in range(0, len(posteriors)):
                print("P(" + node_id + "=" +
                      net.get_outcome_id(node_handle, i) +
                      ")=" + str(posteriors[i]))

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

6.2.3 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);
        }

    }
}
```

6.3 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.

Java:

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

Python:

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

C#:

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

The identfiers of node's outcomes are next.

Java:

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

Python:

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

C#:

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

The parents of the node follow.

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();
}
```

Python:

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

C#:

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

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

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);
```

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)
```

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);
```

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
```

6.3.1 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];
        }
    }

}
```

6.3.2 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("Node id/name: " + net.get_node_id(node_handle) + "/" +
              net.get_node_name(node_handle))
        print("  Outcomes: " + " ".join(net.get_outcome_ids(node_handle)))

        parent_ids = net.get_parent_ids(node_handle)
        if len(parent_ids) > 0:
            print("  Parents: " + " ".join(parent_ids))
        child_ids = net.get_child_ids(node_handle)
        if len(child_ids) > 0:
            print("  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])
            out_str = "    P(" + outcome

            if dim_count > 1:
                out_str += " | "
                for parent_idx in range(0, len(parents)):
                    if parent_idx > 0:
                        out_str += ","
                    parent_handle = parents[parent_idx]
                    out_str += net.get_node_id(parent_handle) + "=" + \
                    net.get_outcome_id(parent_handle, coords[parent_idx])

            prob = cpt[elem_idx]
            out_str += ")=" + str(prob)
            print(out_str)


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

6.3.3 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");
```

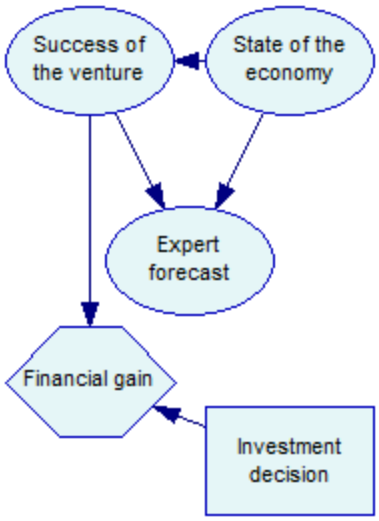
```
    for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
    {
        PrintNodeInfo(net, h);
    }

    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];
            }
        }

    }
}
```

6.4 Tutorial 4: Creating the Influence Diagram

We will further expand the model created in *Tutorial 1* [32] 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 of 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 programs starts by reading the file, just like *Tutorial 2* [41] and *Tutorial 3* [47]. We convert the identfier 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* [32]. 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 x 2 x 1 = 4. The program therefore specifies four numbers for the utilities.

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);
```

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)
```

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);
```

The influence diagram is now complete, so we can write its contents to the file and exit the function. *Tutorial 5* [61] will load the file and perform the inference.

6.4.1 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;
    }

}
```

6.4.2 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
```

6.4.3 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.List,
                    "Invest", "Investment decision",
                    new String[] { "Invest", "DoNotInvest" }, 160, 240);

            int g = CreateNode(net, Network.NodeType.Table,
                    "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;
        }

    }
}
```

6.5 Tutorial 5: Inference in an Influence Diagram

This tutorial loads the influence diagram that we have created in *Tutorial 4* [55]. 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.

Java:

```
double[] expectedUtility = net.getNodeValue("Gain");
int[] utilParents = net.getValueIndexingParents("Gain");
printGainMatrix(net, expectedUtility, utilParents);
```

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)
```

C#:

```
double[] expectedUtility = net.GetNodeValue("Gain");
int[] utilParents = net.GetValueIndexingParents("Gain");
printGainMatrix(net, expectedUtility, utilParents);
```

`printGainMatrix` is a modification of `printCptMatrix` from *Tutorial 3* [47]. 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*[41].

Java:

```
changeEvidenceAndUpdate(net, "Forecast", "Good");
```

Python:

```
self.change_evidence_and_update(net, "Forecast", "Good")
```

C#:

```
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.

6.5.1 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];
        }
    }

}
```

6.5.2 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)
            str_to_print = "    Utility("
            if dim_count > 1:
                for parent_idx in range(0, len(parents)):
                    if parent_idx > 0:
                        str_to_print += ","
                    parent_handle = parents[parent_idx]
                    str_to_print += net.get_node_id(parent_handle) + \
                    "=" + net.get_outcome_id(parent_handle,
                                             coords[parent_idx])
            str_to_print += ")=" + str(mtx[elem_idx])
            print(str_to_print)
        print("")


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

6.5.3 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];
            }
        }

    }
}
```

This page is intentionally left blank.

Appendix: R, rJava and jSMILE

7 Appendix: R, rJava and jSMILE

R programmers can use SMILE's functionality through the rJava package (more information available at *https://CRAN.R-project.org/package=rJava*). rJava provides low-level interface to Java Virtual Machine, effectively making possible calls from R to jSMILE.

To ensure that JVM can find jSMILE's jar file and native library, use `.jinit` parameters:

```
.jinit("<jSMILE jar path>", "-Djsmile.native.library=<native library path>")
```

Alternatively, set the locations later:

```
.jaddClassPath("<jSMILE jar path>")
J("java.lang.System")$setProperty("jsmile.native.library","<native library path>")
```

For more information regarding the native library part of jSMILE, see Java and jSMILE[10].

Here's the R equivalent of the "Hello, SMILE" program, which loads the model from the VentureBN.xsdl file, sets the evidence and displays the calculated posterior probabilties. The code assumes that JVM was started with `.jinit` and SMILE license was initialized with `.jnew("smile.License", ...)` call.

```
net <- .jnew("smile.Network")
.jcall(net,, "readFile", "VentureBN.xdsl")
.jcall(net,, "setEvidence", "Forecast", "Moderate")
.jcall(net,, "updateBeliefs")
beliefs <- .jcall(net, "[D", "getNodeValue", "Success")
for (i in 1 : length(beliefs)) {
    outcomeId = .jcall(net, "S", "getOutcomeId", "Success", as.integer(i-1))
    print(paste(outcomeId, "=", beliefs[i]))
}
```

By default, rJava marshals numeric parameters as doubles to the JVM side. If the jSMILE method to be called requires integer parameters, the `as.integer` function must be called to coerce the value, as in the `getOutcomeId` call inside the for loop. Without as.integer, the code would cause the following rJava error:

```
method getOutcomeId with signature (Ljava/lang/String;D)Ljava/lang/String; not found
```

Since 2nd input parameter of Network.getOutcomeId is an integer, the proper singature is `(Ljava/lang/String;I)Ljava/lang/String;`, the difference is just one letter (I for integer instead of D for double). Using `as.integer` ensures that rJava can correctly infer the signature.

Note how calls to `smile.Network` method returning `void` do not require the declaration of the return type (the default value of the second `.jcall` parameter is used). On the other hand, to retrieve the posteriors with `smile.getNodeValue`, its return type, which is `double[]` in Java, must be passed to `.jcall`. rJava uses Java Native Interface (JNI) type signatures and `double[]` becomes `"[D"`. For convenience, rJava treats `"S"` as signature for Java's strings (the full signature is `"Ljava/lang/String;"`) - the `smile.Network.getOutcomeId` invocation inside the for loop takes advantage of the

shorthand notation. The complete list of type signatures is included in the JNI documentation at *https://docs.oracle.com/javase/9/docs/specs/jni/types.html#type-signatures*

This page is intentionally left blank.

Acknowledgments

8 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

Expat

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) HOWEVE 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.

This page is intentionally left blank.

- A -

- D -

- L -