AMiDST	FP7-ICT 619209 / AMIDST 24/02/2015 Page 1 of …	

Project no.: 619209

Project full title: Analysis of Massive Data STreams

Project Acronym: AMIDST

Deliverable no.: D2.3

Title of the deliverable: A software library implementation of the modelling framework

Contractual Date of Delivery to the CEC:	**31.03.2015**
Actual Date of Delivery to the CEC:	**31.03.2015**
Organisation name of lead contractor for this deliverable:	**AAU**
Author(s):	**Hanen Borchani, Antonio Fernández, Helge Langseth, Anders L. Madsen, Ana M. Martínez, Andrés Masegosa, Thomas D. Nielsen, Antonio Salmerón**
Participants(s):	**P01, P02, P03, P04, P05, P06, P07**
Work package contributing to the deliverable:	**WP2**
Nature:	**R**
Version:	**1.0**
Total number of pages:	**...**
Start date of project:	**1st January 2014 Duration: 36 month**

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

In this document, we describe the software library implementation of the AMIDST modelling framework.

Keyword list: AMIDST modelling framework, software library, implementation.

Contents

Document history

Version	Date	Author (Unit)	Description
v0.3	10/03/2015	All consortium members	The software library implementation for the AMIDST modelling framework discussed and established
v0.6	20/03/2015	Hanen Borchani, Antonio Fernández, Helge Langseth, Anders L. Madsen, Ana M. Martínez, Andrés Masegosa, Thomas D. Nielsen, Antonio Salmerón	Initial version of document finished and reviewed
v1.0	31/03/2015	Hanen Borchani, Antonio Fernández, Helge Langseth, Anders L. Madsen, Ana M. Martínez, Andrés Masegosa, Thomas D. Nielsen, Antonio Salmerón	Final version of document

1 Executive summary

The aim of this document is to describe the main features of the software library implementation of the AMIDST modelling framework, namely, the data structure and the core components.

The software library design and the status of the toolbox implementation have been already introduced in Section 4, Deliverable 4.1 [1], basically covering AMIDST software learning capabilities. In this deliverable, we provide an update of the status of the toolbox implementation as well as an extended overview of the AMIDST software library.

2 Introduction

In this deliverable, we present an overview of the software library implementation of the AMIDST toolbox.

Section 3 gives a general overview of the core components of the framework, including data structures for variables, graphs, Bayesian networks, dynamic Bayesian networks, key distributions such as multinomial and conditional linear Gaussian distributions represented in both standard form and as exponential families. Section provides a description of the considered database functionalities that will be used by AMIDST learning and inference algorithms. Section 5 presents the functionalities defined for transforming AMIDST models to and from Hugin API.

3 Data structures

An overview of the different data structures of the AMIDST toolbox is illustrated in Figure 3.1. These data structures basically define the main components that are used afterwards for implementing the AMIDST learning and inference algorithms. In what follows, we briefly define each component showing how it can be used through providing some code excerpts.

3.1 Probabilistic graphical model (PGM)

A probabilistic graphical model is a framework consisting of two parts: a qualitative component in the form of a graphical model encoding conditional independence assertions about the domain being modelled as well as a quantitative component consisting of a collection of local probability distributions adhering to the independence properties specified in the graph- ical model. Collectively, the two components provide a compact representation of the joint probability distribution over the domain being modelled.

In the AMIDST toolbox, we currently focus on two specific instantiations of PGMs, namely, a static Bayesian network (BN component) and a two time-slice dynamic Bayesian network (2T-DBN).

3.2 Static variables

Static variables consist of a list of objects of type `Variable` that are used later to build a static Bayesian network. Each static variable is characterized by its name, ID, the state space type, the distribution type (i.e., multinomial or normal), as well as if it is observed or not.

Note that observed static variables are initialised using the list of attributes (that are

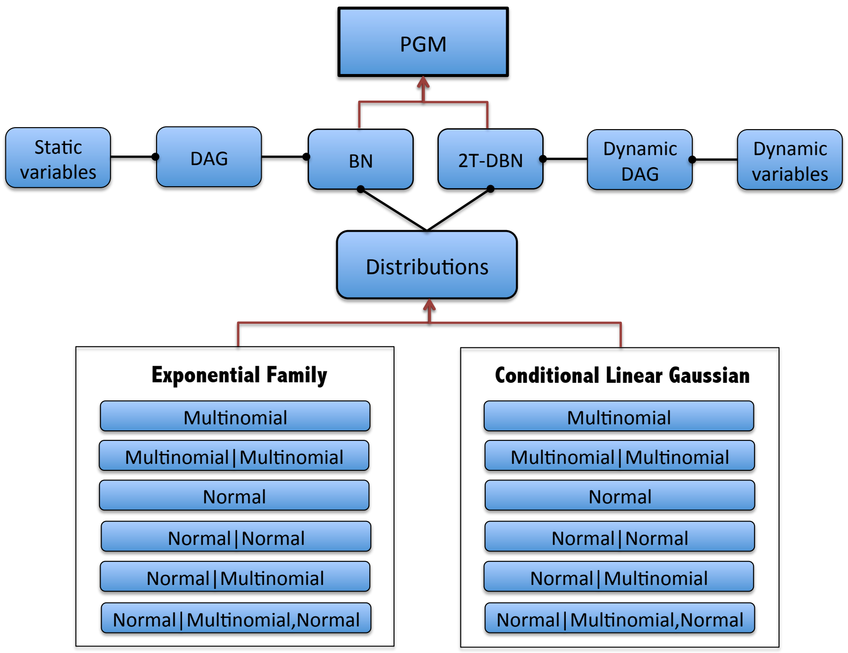

Figure 3.1: Illustration of AMIDST toolbox data structure components. Nomenclature: The boxes in the figure represent software components (sets, possibly singletons, of classes), a rounded-arc going from X to Y indicates that Y 'uses/references' X, and an arc with an arrow from X to Y implies inheritance.

already parsed from the dataset or specified by the user), then hidden static variables are afterwards specified by the user.

```
StaticVariables variables = new StaticVariables(data.getAttributes());
Variable A = variables.getVariableByName("A");
Variable B = variables.getVariableByName("B");
Variable C = variables.getVariableByName("C");

VariableBuilder variableBuilder = new VariableBuilder();
variableBuilder.setName("HiddenVar");
variableBuilder.setObservable(false);
variableBuilder.setStateSpace(new
      MultinomialStateSpace(Arrays.asList("TRUE","FALSE")));
variableBuilder.setDistributionType(DistType.MULTINOMIAL);
Variable hidden = variables.addHiddenVariable(variableBuilder);
```

3.3 Directed acyclic graph (DAG)

A directed acyclic graph (DAG) defines the Bayesian network graphical structure over a list of Static variables, such that he dependence relationships between the variables are established through the definition of the parent set for each variable.

```
WekaDataFileReader reader = new
      WekaDataFileReader("data/dataWeka/contact-lenses.arff");
StaticVariables variables = new StaticVariables(reader.getAttributes());
DAG dag = new DAG(variables);

StaticVariables variables = dag.getStaticVariables();
Variable A = variables.getVariableById(0);
Variable B = variables.getVariableById(1);
Variable C = variables.getVariableById(2);
Variable D = variables.getVariableById(3);

dag.getParentSet(B).addParent(A);
dag.getParentSet(C).addParent(A);
dag.getParentSet(D).addParent(B);
dag.getParentSet(D).addParent(C);
```

3.4 Bayesian network (BN)

A static Bayesian network consists of two components: a graphical structure (defined by the DAG component) and conditional probability distributions of each variable given the set of its parents (defined by the Distributions component).

The distribution of each variable in the Bayesian network is initialised and specified according to its type and the type of its potential parent set. After this step, the set of parents of each variable becomes unmodifiable.

This is brief code fragment showing the definition of a Bayesian network using the previously created `dag`. It automatically looks at the distribution type of each variable and their parents to initialise the Distributions objects that are stored inside (i.e., Multinomial, Normal, CLG, etc). The parameters defining these distributions are correspondingly initialised.

```
BayesianNetwork bn = BayesianNetwork.newBayesianNetwork(dag);
```

3.5 Dynamic variables

Dynamic variables consist of a list of objects named allVariables and temporalClones of type `Variable`, that are used to build dynamic Bayesian networks. Each dynamic variable is characterized by its name, ID, the state space type, the distribution type (i.e., multinomial or normal), and if it is observed or not. In order to represent the variables in a previous time step (needed when defining the dynamic DAG), we use the concept of *temporal clone* variables, which are copies of the real main variables but refer to the previous time step. For instance, X_{t-1} is codified as the *temporal clone* of variable X_t. Hence, in our data structures, the time index t is not explicitly represented for a dynamic variable, but implicitly considered with the use of *temporal clones*.

The list of observable dynamic variables and their temporal clones is initialised using the list of Attributes (that are already parsed from the dataset or specified by the user), then hidden variables and their temporal clones can be also added by the user.

```
DynamicVariables dynamicVariables = new DynamicVariables();
Variable observedROP = dynamicVariables.addObservedDynamicVariable(attROP);
Variable observedTRQ = dynamicVariables.addObservedDynamicVariable(attTRQ);
Variable realTRQ = dynamicVariables.addRealDynamicVariable(observedTRQ);
VariableBuilder variableBuilder = new VariableBuilder();
variableBuilder.setName("HiddenVar");
variableBuilder.setObservable(false);
variableBuilder.setStateSpace(new RealStateSpace());
variableBuilder.setDistributionType(DistType.GAUSSIAN);
Variable hidden = dynamicVariables.addHiddenDynamicVariable(variableBuilder);
```

3.6 Dynamic directed acyclic graph (Dynamic DAG)

A dynamic directed acyclic graph (Dynamic DAG) defined over a list of dynamic variables. This component specifies the graph structure of a 2T-DBN, i.e., the parent set for each dynamic variable at both time 0 and at time $t > 0$.

```
DynamicDAG dynamicDAG = new DynamicDAG(dynamicVariables);

dynamicDAG.getParentSetTimeT(observedTRQ).addParent(observedWOB);
dynamicDAG.getParentSetTimeT(observedTRQ).addParent(observedRPMB);
dynamicDAG.getParentSetTimeT(observedTRQ).addParent(observedMFI);
dynamicDAG.getParentSetTimeT(observedTRQ).addParent(realTRQ);
dynamicDAG.getParentSetTimeT(observedTRQ).addParent(hidden);
dynamicDAG.getParentSetTimeT(observedTRQ).addParent(mixture);
```

3.7 Two time-slice dynamic Bayesian network (2T-DBN)

Similarly to a BN, a 2T-DBN (see Deliverable D2.1, Section 3.4 [2]) is defined using two main components: a graphical structure (defined by the Dynamic DAG component) and conditional probability distributions of each dynamic variable given the set of its parents (defined by the Distributions component). The distributions of each dynamic variable at both time 0 and time T are initialised and specified according to the variable type and the type of its potential parent set. After this step, the set of parents of each dynamic variable becomes unmodifiable.

This is brief code fragment showing the definition of a dynamic Bayesian network using the previously created `dynamicDAG`. It automatically looks at the distribution type of each variable and their parents to initialise the Distributions objects that are stored inside (i.e., Multinomial, Normal, CLG, etc). The parameters defining these distributions are correspondingly initialised.

```
DynamicBayesianNetwork dynamicBayesianNetwork =
 DynamicBayesianNetwork.newDynamicBayesianNetwork(dynamicDAG);
```

3.8 Distributions

The Distributions component consists of the set of conditional probability distributions considered in the AMIDST toolbox, including variables with both multinomial and normal distributions.

Note here that, in spite of the distinction between BN and 2T-BN, the distributions over both models could be defined in the same way, and thereby the parameter learning and inference algorithms could be also applied equally for both models. In particular, the Distributions component includes the set of conditional probability distributions considered in the AMIDST toolbox (the so-called Conditional Linear Gaussian distributions, as detailed in Deliverable 2.1 [2]). More precisely, both variables with multinomial and normal distributions are modeled, and the distribution of each variable, in either a BN or 2T-BN, is initialized and specified according to its distribution type and the distribution types of its potential parents. This consequently gives rise to the following different implemented probability distributions:

- Multinomial: a multinomial variable with no parents.
- Multinomial|Multinomial: a multinomial variable with multinomial parents.
- Normal: a normal variable with no parents.
- Normal|Normal: a normal variable with normal parents.
- Normal|Multinomial: a normal variable with multinomial parents.
- Normal|Multinomial,Normal: a normal variable with a mixture of multinomial and normal parents.

The case of a multinomial variable having normal parents is not considered yet in this initial prototype. It is planned to be included in future versions, although strongly restricted in inference and learning algorithms due to the methodological and computational issues previously commented in Deliverable D2.1 [2].

We also provide an implementation of all the above distributions in the so-called Exponential Family form, which ensures an alternative representation of the standard distributions based on vectors of natural and moment parameters.

The following brief code fragment shows the definition of the distribution for a variable `var` given the set of its parents:

```
ParentSet parentSet = this.getDAG().getParentSet(var);
int varID = var.getVarID();
this.distributions[varID]=
     DistributionBuilder.newDistribution(var, parentSet.getParents());
parentSet.blockParents();
```

4 Database management

This section covers the description of databases that will be used by AMIDST learning and inference algorithms implemented in the toolbox. Figure 4.1 shows a high-level overview of the key components of the AMIDST software tool. It illustrates mainly the different database functionalities and how they are connected to the core component PGM through both the Learning Engine and Inference Engine components. In what follows, we describe each of the database functionalities, along with a code excerpt containing a brief example how to define the described functionality., then introduce briefly the Learning Engine and Inference Engine that will be presented in more details in Deliverable 3.2.

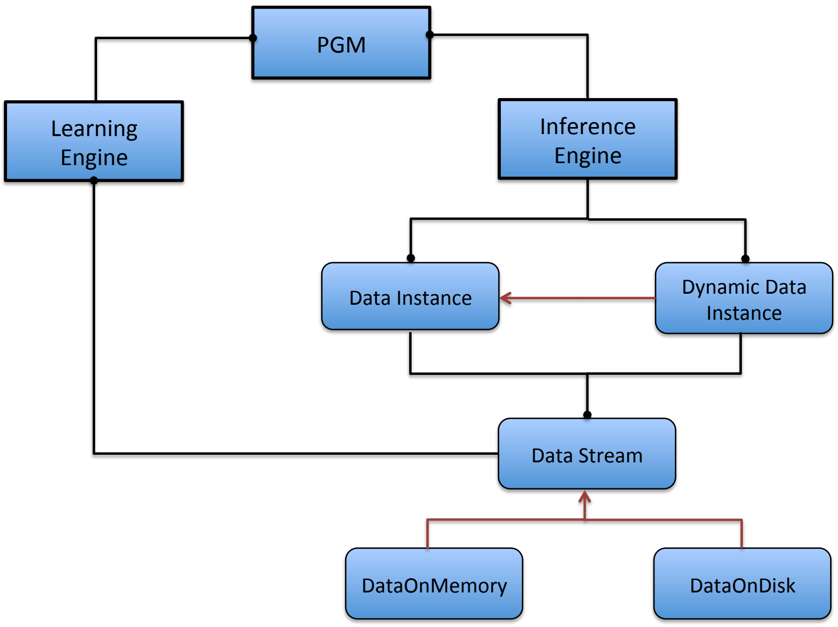

Figure 4.1: Illustration of the main database management functionalities and their connection with PGM, learning engine, and inference engine components.

4.1 DataStream

In the AMIDST framework, we consider a streaming data as a data source, where data arrives at high frequency with no storage of historical data. Then, from this general (Data Stream) component, the specialised database functionalities could be derived, namely, (DataOnMemory and DataOnDisk.

In addition, Data Stream is connected to Data Instance and Dynamic Data Instance components.

The employed design is intended to support future users and developers of the AMIDST toolbox in the potential design and implementation of other database specifications; the only restriction being that new database components should implement the interface defined by the Data Stream component.

4.2 DataOnMemory

DataOnMemory implements database functionality for data sets that can be loaded into main memory.

4.3 DataOnDisk

DataOnDisk provides functionality for handling datasets too large to be loaded in main memory.

4.4 DataInstance

The Data Instance component consists of a single class that can represent a particular evidence configuration, such as the observed values of a collection of variables at time t or a particular row in a database.

4.5 DynamicDataInstance

The DynamicDataInstance always has a TimeID and a SequenceID. If this two attributes, or any of the two, are not in the dynamic data set, then they are automatically filled in, incrementally for the TimeID and with a value of 1 for the SequenceID.

4.6 Learning Engine

Implementations of learning algorithms will be provided through Learning Engine component for static and dynamic Bayesian networks, ensuring both the structural and parameter learning.

For structural learning, the AMIDST toolbox design includes components for supporting PC and TAN learning in a parallel setting (cf. Task 4.1). The current implementation supports standard PC learning and parallel TAN learning by interfacing to the Hugin API.

For parameter learning, a fully Bayesian approach is pursued in the AMIDST framework (cf. the activities in Task 4.2 and Task 4.4). This, in turn, means that parameter learning reduces to the task of inference for which we plan to consider two approaches: variational message passing and expectation propagation. More implementation details about these two algorithms will be provided in Deliverable 3.2.

Note that the design of the learning engine of the AMIDST framework is flexible in the sense that it easily accommodates potential future learning-based extensions of the framework, e.g., Bayesian learning based on importance sampling or maximum likelihood learning using the expectation maximization algorithm (see Section 3 in Deliverable 4.1 [1]).

4.7 Inference Engine

As previously noted in Deliverable 4.1 [1] (see Section 3), efficient implementations of both variational message passing and expectation propagation algorithms can be realized when the distribution families of the models are conjugate-exponential. In this case, the inference operations can be further supported by specifying the exponential distributions using their natural parameters.

These functionalities are ensured in AMIDST toolbox through tailored exponential family implementations of the standard distributions that are part of the AMIDST framework (such as the conditional linear Gaussian distribution).

5 Hugin link

This component includes all the functionalities needed to link the AMIDST toolbox with the Hugin software. This connection is primarily ensured by converting Hugin models into AMIDST models, and vice versa.

This is extremely useful as it allows us for instance to test and assess some of the implemented AMIDST functionalities within a well-stablished platform as Hugin. For instance, a new inference algorithm implemented in AMIDST could be compared with some state-of-the-art algorithms included in Hugin. In addition, the connection with Hugin could be more efficient as it extends and provides some extra functionalities to AMIDST toolbox, such as the use of parallel TAN for structural learning.

5.1 BN converter from AMIDST to Hugin format

This functionality addresses the conversion of a Bayesian network from AMIDST to Hugin. This conversion is done at “object-level”, which is far more efficient that if done by converting the models to data files and, then, parsing them.

This is brief code fragment showing how to convert a BN from Amidst to Hugin format and stored it on a file. Then, we can open HUGIN and visually inspect the BN created with the AMIDST toolbox.

```
BayesianNetwork amidstBN = BayesianNetwork.newBayesianNetwork(dag);
Domain huginNetwork = ConverterToHugin.convertToHugin(amidstBN);
huginNetwork.saveAsNet("networks/huginStaticBNHiddenExample.net");
```

5.2 BN converter from Hugin to AMIDST format

This functionality addresses the conversion of a Bayesian network from Hugin to Amidst. This conversion is done at "object-level", which is far more efficient that if done by converting the models to data files and, then, parsing them.

This is brief code fragment showing how to convert a BN from Hugin to Amidst.

```
ParseListener parseListener = new DefaultClassParseListener();
Domain huginBN = new Domain ("networks/huginNetwork.net", parseListener);
BayesianNetwork amidstBN = ConverterToAMIDST.convertToAmidst(huginBN);
```

6 Conclusion

References

[1] Borchani, H., Langseth, H., Nielsen, T.D., Salmerón, A., Madsen, A.L.: Progress report on the software development (December 2014)

[2] Borchani, H., Fernández, A., Gundersen, O.E., Hovda, S., Langseth, H., Madsen, A.L., Martínez, A.M., Martínez, R.S., Masegosa, A., Nielsen, T.D., Salmerón, A., rmo, F.S., Weidl, G.: The AMIDST modelling framework – Initial draft report (September 2014) Deliverable 2.1 of the AMIDST project, amidst.eu.