Project no.: 619209

Project full title: Analysis of Massive Data STreams

Project Acronym: AMIDST

Deliverable no.: D3.3

Title of the deliverable: Prototype software modules that will be integrated into the general AMIDST prototype library.

Contractual Date of Delivery to the CEC:	**30.11.2015**
Actual Date of Delivery to the CEC:	**30.11.2015**
Organisation name of lead contractor for this deliverable:	**AAU**
Author(s):	**Hanen Borchani, Helge Langseth, Anders L. Madsen, Ana M. Martínez, Andrés Masegosa, Thomas D. Nielsen, Darío Ramos López, Antonio Salmerón**
Participants(s):	**P01, P02, P03, P04**
Work package contributing to the deliverable:	**WP3**
Nature:	**P**
Version:	**1.0**
Total number of pages:	**28**
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 prototype software modules related to inference integrated into the AMIDST toolbox. In particular, we describe the modules related to the (static) Inference Engine (already discussed in D3.2), focusing on the inclusion of the Maximum A Posteriori (MAP) and Most Probable Explanation (MPE) inference algorithms; the most recently developed Dynamic Inference Engine of the AMIDST toolbox, which includes a complete operative implementation of the dynamic MAP; and dynamic variational message passing and importance sampling algorithms using the Factored Frontier Algorithm to bypass the entanglement problem. Class diagrams are included for the inference packages.

Keyword list: AMIDST software development, AMIDST toolbox, Dynamic Inference Engine, Most Probable Explanation (MPE), Maximum A Posteriori (MAP), variational message passing (VMP).

Contents

Document history

Version	Date	Author (Unit)	Description
v0.3	12/11/2015	Hanen Borchani, Helge Langseth, Anders L. Madsen, Ana M. Martínez, Andrés Masegosa, Thomas D. Nielsen, Darío Ramos López, Antonio Salmerón	Content of the document discussed and established
v0.6	26/11/2015	Hanen Borchani, Helge Langseth, Anders L. Madsen, Ana M. Martínez, Andrés Masegosa, Thomas D. Nielsen, Darío Ramos López, Antonio Salmerón	Initial version of document finished and reviewed by the PSRG
v1.0	27/11/2015	Hanen Borchani, Helge Langseth, Anders L. Madsen, Ana M. Martínez, Andrés Masegosa, Thomas D. Nielsen, Darío Ramos López, Antonio Salmerón	Final version of document

1 Executive summary

The aim of this document is to describe the prototype software modules that will be integrated into the general AMIDST prototype library related to inference. This deliverable complements Deliverable D3.2 [1], where the progress of the software development related to inference in the AMIDST project was reported in Month 15. Likewise, this development is based on the AMIDST modelling framework (cf. Deliverable 2.1 [2] and Deliverable 2.2 [3]) as well as the methodological study of the probabilistic inference problem in hybrid and dynamic models (cf. Deliverable 3.1 [4] and Deliverable 3.4).

The module of the AMIDST toolbox implementing the inference functionality for static models is referred to as the Inference Engine. Its dynamic counterpart is developed in the Dynamic Inference Engine module.

The Inference Engine includes complete operative implementations of the variational message passing (VMP), importance sampling (IS), maximum a posteriori (MAP) and most probable explanation (MPE) algorithms. It also supports exact inference through the AMIDST HUGIN API interface that connects the AMIDST toolbox with the HUGIN software. The Dynamic Inference Engine includes dynamic versions of most of these inference algorithms. On one hand, VMP and IS are adapted for dynamic models by means of the Factored Frontier algorithm; whereas Dynamic MAP is implemented by computing (variational) posterior distributions over subsets of MAP variables.

2 Introduction

This document provides a description of the prototype software modules that constitute the inference part of the AMIDST toolbox. It relates to WP3, which corresponds to exact and approximate inference in hybrid and dynamic Bayesian networks.

In particular, we describe the developed Inference Engine and Dynamic Inference Engine modules of the AMIDST software. The Inference Engine includes a complete operative implementation of the variational message passing [5], importance sampling (IS) [6], and two abductive inference algorithms [7], namely maximum a posteriori (MAP) and most probable explanation (MPE). Exact inference is supported through the defined interface AMIDST HUGIN API, connecting the AMIDST software with the HUGIN software.

The Dynamic Inference Engine module includes dynamic versions of the VMP and IS algorithms that make use of the Factored Frontier (FF) algorithm [8] to avoid the entanglement problem. Additionally, this module support dynamic MAP inference using a novel technique that computes the (variational) posterior distributions over different subsets of MAP variables.

The structure of the document is as follows: Section 3 briefly introduces the required background related to the considered models and the probabilistic inference problem. Section 4 provides an overview of the inference module in AMIDST and its connection with other modules in the toolbox. Next, Section 5 and 6 contain a description of the implemented Inference Engine and Dynamic Inference Engine modules of the AMIDST toolbox. Finally, we conclude in Section 7. An appendix with all the relevant class diagrams is included at the end of the document.

3 Preliminaries

Bayesian networks (BNs) [9, 10] are a particular type of probabilistic graphical model (PGM) that has enjoyed widespread attention during the last two decades. In general, for a BN with n variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, the joint probability distribution factorises as $p(\mathbf{X}) = \prod_{i=1}^{n} p(X_i|\text{pa}\,(X_i))$, where $\text{pa}\,(X_i)$ denotes the parent set of X_i in the network.

A BN is called *hybrid* if some of its variables are discrete while others are continuous. A BN is called *dynamic* [11, 12] if it models a domain that evolves over time by encoding explicitly the temporal dynamics of the system. In particular, a *two-time slice dynamic BN* (2T-DBN), with n variables $\mathbf{X}^t = \{X_1^t, \ldots, X_n^t\}$, is defined by an *initial model* representing the initial joint distribution of the process at time 0, and a *transition model* representing a standard BN repeated over time, i.e., at time $t+1$ (see [13, Section 2] for an extended description of BNs and 2T-DBNs).

Given an already specified model, the problem of *probabilistic inference* consists of computing the posterior distribution over the values of some variables of interest given the observed values of other variables (also know as evidence). Formally, let $\mathbf{X}_E \subset \mathbf{X}$ de-

note a set of observed variables, and $\mathbf{X}_I \subseteq \mathbf{X} \setminus \mathbf{X}_E$ denote a set of variables of interest. *Probabilistic inference* consists of computing $p(x_i|\mathbf{x}_E)$ for each $i \in I$, where x_i is a value of X_i, and $\mathbf{x}_E$ is a configuration of the variables in $\mathbf{X}_E$.

A particularly complex kind of inference in BNs is the so-called maximum a posteriori (MAP) problem. For a set of target variables $\mathbf{X}_I \subseteq \mathbf{X} \setminus \mathbf{X}_E$, the goal of MAP inference is to compute

$$\mathbf{x}_I^* = \arg\max_{\mathbf{x}_I \in \Omega_{\mathbf{X}_I}} p(\mathbf{x}_I|\mathbf{X}_E = \mathbf{x}_E),$$

where $p(\mathbf{x}_I|\mathbf{X}_E = \mathbf{x}_E)$ is obtained by first marginalising out from the joint distribution $p(\mathbf{x})$ the variables not in $\mathbf{X}_I$ and not in $\mathbf{X}_E$ and where $\Omega_{\mathbf{X}_I}$ stands for the set of possible values in $\mathbf{X}_I$.

A related problem is MPE where the objective is to find the most probable explanation to an observation $\mathbf{X}_E = \mathbf{x}_E$. It is a particular case of MAP, where $\mathbf{X}_I = \mathbf{X} \setminus \mathbf{X}_E$. Both MAP and MPE belong to the class of problems known as abductive inference [7].

Several exact and approximate inference algorithms have been proposed. In Deliverable 3.1 [4] we presented a thorough review of the state-of-the-art of probabilistic inference in hybrid and dynamic BNs. For hybrid BNs, exact and approximate inference algorithms are categorised into three groups: 1) algorithms based on the Gaussian assumption, 2) algorithms that operate over general densities, i.e., they do not rely on the Gaussian assumption, and 3) algorithms based on a transformation of the original densities. More details about these algorithms and their references can be found in [4, Section 3]. Moreover, for dynamic BNs, approximate methods including approximate factorisations of the joint probability distribution, sampling based techniques, and variational approximations are discussed in [4, Section 5]. Deliverable 3.4 [14] includes a description of more efficient inference algorithms in hybrid domains for both static and dynamic models.

In the AMIDST toolbox we can find two separate modules related to inference. The Inference Engine, already introduced in Deliverable 3.2 [1], includes all inference algorithms that can be applied in (static) Bayesian networks. Specifically, it includes five different approximate inference algorithms: one based on variational inference [15, 16], namely, variational message passing [5]; an importance sampling-based algorithm [6]; an exact inference algorithm developed in the HUGIN software; and two abductive inference algorithms [7], namely, most probable explanation and maximum a posteriori inference algorithms. The Dynamic Inference Engine includes a dynamic version of the VMP and IS inference algorithms using the factored-frontier algorithm [8] and a dynamic MAP approach. The two modules are designed to easily accommodate any type of inference algorithm.

4 Inference in AMIDST: an overview

In this section, we provide an overview of the different software modules in the AMIDST toolbox and its connection with the Inference Engine.

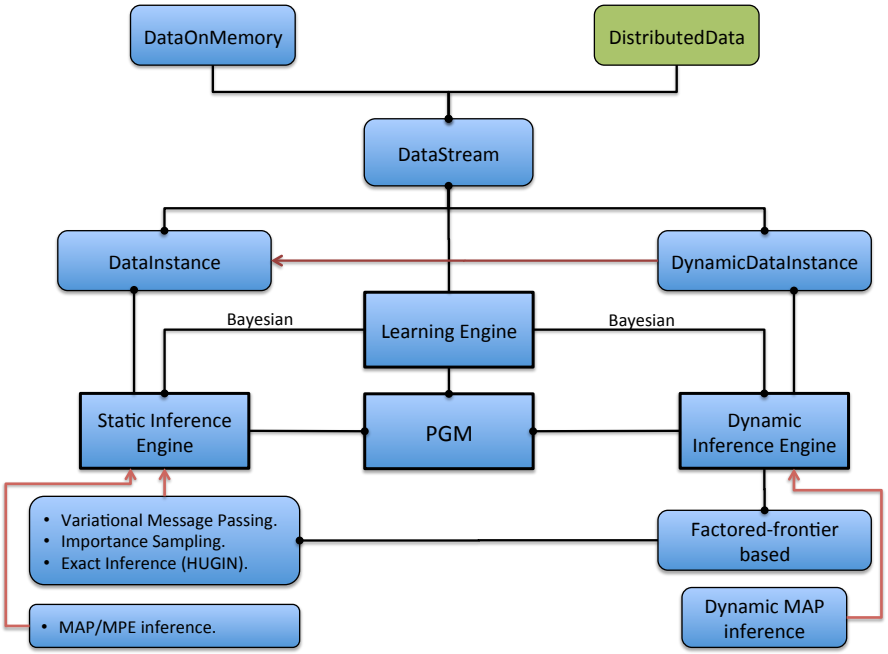


Figure 4.1: Illustration of the core components of the AMIDST toolbox, making focus on the Inference modules.

Figures 4.1 illustrates the main core components of the AMIDST toolbox, more specifically:

- the considered data source management functionalities, namely, DataOnMemory, DistributedData, DataStream, DataInstance, and DynamicDataInstance (see [13, Section 6]). The green box for the DistributedData component indicates that this module corresponds to work in progress that will be reported in the remaining deliverables for WP4.
- the Learning Engine component, including structure and parameter learning of static and dynamic BNs (note that the progress of the software development related to learning is described in Deliverable 4.1 [17]).
- the Inference Engine component is shown as divided into a Static Inference Engine for static Bayesian networks and a Dynamic Inference Engine component for dynamic

Bayesian networks. The following sections include more details about the inference algorithms in these two components.

It is possible to see in this diagram the existing close link between the learning and the inference algorithms when a Bayesian approach is used for learning, that is, for variational message passing algorithms.

5 Static Inference Engine

The Inference Engine component includes all the inference algorithms for (static) Bayesian networks. It consists of five different methods, namely, Variational Message Passing, Importance Sampling, HUGIN Exact Inference, MPE Inference and MAP Inference. This module has been designed and implemented to be extendable and support future implementations of other inference algorithms. Please see the appendix for the class diagrams for the inference packages.

5.1 Variational Message Passing (VMP)

The first inference algorithm implemented in the AMIDST toolbox is variational message passing (VMP) [5]. VMP optimises a variational bound using a set of local computations for each node, together with a mechanism for passing messages between the nodes. The messages belong to exponential family of distributions [5, 16, 18], summarised either by their natural parameter vector (for child-to-parent messages) or by a vector of moments (for parent-to-child messages). These messages are defined so that the optimal variational distribution for a node can be found by summing the messages from its children together with a function of the messages from its parents, where this function depends on the conditional distribution for the node in question [5].

The code in Listing 1 shows an example of use of the VMP inference algorithm.

Listing 1: Example of Variational Message Passing

```
public static void main(String[] args) throws Exception {

    //We load the WasteIncinerator BN, contains multinomial and Gaussian variables.
    BayesianNetwork bn = BayesianNetworkLoader.loadFromFile("./networks/
        WasteIncinerator.bn");

    //We recover e.g. two relevant variables: Mout (Gaussian) and W (multinomial).
    Variable varMout = bn.getVariables().getVariableByName("Mout");
    Variable varW = bn.getVariables().getVariableByName("W");

    //We create an instance of a VMP inference algorithm.
    InferenceAlgorithm inferenceAlgorithm = new VMP();
    //Then, we set the BN model
    inferenceAlgorithm.setModel(bn);
```

```
//Set the evidence.
Assignment assignment = new HashMapAssignment(1);
assignment.setValue(varW,0);
inferenceAlgorithm.setEvidence(assignment);

//Then we run inference
inferenceAlgorithm.runInference();

//We can query the posterior of
System.out.println("P(Mout|W=0) = " + inferenceAlgorithm.getPosterior(varMout));

//Or some more refined queries
System.out.println("P(0.7<Mout<6.59 | W=0) = " + inferenceAlgorithm.
    getExpectedValue(varMout, v -> (0.7 < v && v < 6.59) ? 1.0 : 0.0 ));

//We can also compute the probability of the evidence
System.out.println("P(W=0) = "+Math.exp(inferenceAlgorithm.
    getLogProbabilityOfEvidence()));
}
```

5.2 Importance Sampling (IS)

Importance Sampling (IS) [19] is a Monte Carlo technique based on transforming the parameters to estimate into expected values with respect to a so-called *sampling distribution*, that are subsequently estimated through a weighted sample mean.

Inference based on IS is a three-step procedure that involves the following three tasks:

1. Selecting a sampling distribution.
2. Drawing a sample from the selected sampling distribution.
3. Estimating the parameters of the posterior distribution from the sample.

Different choices of the sampling distribution result in different accuracy levels. The design of IS in the AMIDST toolbox is modular and flexible enough as to allow the implementation of new methods for obtaining the sampling distributions in an easy way. The current version of the implementation is able to get the sampling distributions either directly from the conditional distributions in the Bayesian network (which is equivalent to forward sampling) or to use other methods that provide a valid distribution for each of the variables.

The sampling task has been implemented using multithreading, taking advantage of the fact that the items in the sample are independent and can therefore be generated in parallel. Hence, this part of the implementation is scalable according to the number of cores available for computation. A Map/Reduce parallelization schema can be seen in

Figure 5.2. If the parallel mode is activated, both sampling and the computation of the sufficient statistics are carried out in parallel using all available nodes.

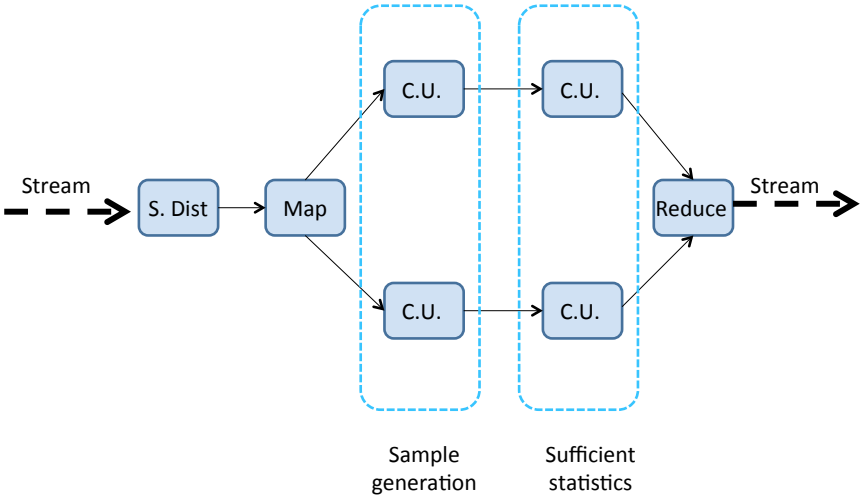


Figure 5.1: Illustration of the Map/Reduce scheme for Importance Sampling.

The code in Listing 2 shows an example of use of the IS inference algorithm. More details about the classes and methods related to this inference method in the AMIDST Toolbox can be found in the appendix.

Listing 2: Example of Importance Sampling

```
public static void main(String[] args) throws Exception {

    //We load the WasteIncinerator BN, contains multinomial and Gaussian variables.
    BayesianNetwork bn = BayesianNetworkLoader.loadFromFile("./networks/
        WasteIncinerator.bn");

    //We recover e.g. two relevant variables: Mout (Gaussian) and W (multinomial).
    Variable varMout = bn.getVariables().getVariableByName("Mout");
    Variable varW = bn.getVariables().getVariableByName("W");

    //First we create an instance of an IS inference algorithm.
    ImportanceSampling inferenceAlgorithm = new ImportanceSampling();

    //Then, we set the BN model
    inferenceAlgorithm.setModel(bn);
```

```
        //Set the evidence.
        Assignment assignment = new HashMapAssignment(1);
        assignment.setValue(varW,0);
        inferenceAlgorithm.setEvidence(assignment);

        //We can also set to be run in parallel on multicore CPUs
        inferenceAlgorithm.setParallelMode(true);

        //To perform more than one operation, data should be kept in memory
        inferenceAlgorithm.setKeepDataOnMemory(true);

        //Then we run inference
        inferenceAlgorithm.runInference();

        //Then we query the posterior of
        System.out.println("P(Mout|W=0) = " + inferenceAlgorithm.getPosterior(varMout));

        //Or some more refined queries
        System.out.println("P(0.7<Mout<6.59 | W=0) = " + inferenceAlgorithm.
            getExpectedValue(varMout, v -> (0.7 < v && v < 6.59) ? 1.0 : 0.0 ));

        //We can also compute the probability of the evidence
        System.out.println("P(W=0) = "+Math.exp(inferenceAlgorithm.
            getLogProbabilityOfEvidence()));
    }
```

5.3 HUGIN Exact Inference

In addition to the above-mentioned approximate inference algorithms, the AMIDST toolbox also supports exact inference for BNs and DBNs by means of the AMIDST HUGIN API interface. The interface allows the AMIDST toolbox to access the exact inference functionality available in HUGIN, which is based on the junction tree algorithm [20].

The code in Listing 3 shows an example of use of this algorithm.

Listing 3: Example of HUGIN Exact Inference

```
    public static void main(String[] args) throws Exception {

        //We load the WasteIncinerator BN, contains multinomial and Gaussian variables.
        BayesianNetwork bn = BayesianNetworkLoader.loadFromFile("./networks/
            WasteIncinerator.bn");

        //We recover e.g. two relevant variables: Mout (Gaussian) and W (multinomial).
        Variable varMout = bn.getVariables().getVariableByName("Mout");
        Variable varW = bn.getVariables().getVariableByName("W");

        //Create an instance of a HUGIN inference algorithm.
        InferenceAlgorithm inferenceAlgorithm = new HuginInference();
        //Then, we set the BN model
```

```
        inferenceAlgorithm.setModel(bn);

        //If exists, set the evidence.
        Assignment assignment = new HashMapAssignment(1);
        assignment.setValue(varW,0);
        inferenceAlgorithm.setEvidence(assignment);

        //Run inference.
        inferenceAlgorithm.runInference();

        //Query the posterior of
        System.out.println("P(Mout|W=0) = " + inferenceAlgorithm.getPosterior(varMout));

        //Or some more refined queries
        System.out.println("P(0.7<Mout<3.5 | W=0) = " + inferenceAlgorithm.
            getExpectedValue(varMout, v -> (0.7 < v && v < 3.5) ? 1.0 : 0.0 ));
    }
```

5.4 MAP and MPE Inference

MAP (*maximum a posteriori*) inference methods are a particular case of inference algorithms that finds the maximum a posteriori configuration over a set of *query* variables given observations of some of the other variables.

MPE (*most probable explanation*) methods are a subtype of the MAP inference techniques, in which all of the non-observed variables are queried.

In the context of the conditional linear Gaussian (CLG) family of networks, the presence of observed continuous variables with unobserved continuous parents can entail a challenge for MPE, and it is often necessary to resort to approximate methods for efficiency reasons.

A simple approximate procedure consists in drawing a big sample from the network, according to the evidence, and picking the configuration that gives the highest joint probability (sampling method).

By applying well-known Monte-Carlo optimization algorithms (such as simulated annealing [21] or hill-climbing [22]) over the discrete variables, and by simulating, according to their conditional distribution and the values assigned to their ancestors (already set if we are following the topological order of the BN), we can perform MPE inference on a CLG Bayesian network efficiently [23]. Additionally, these two optimization methods can operate in either a local or a global mode, the former generates neighbour candidates by shifting only some of the variables, whereas the latter shifts all of them.

Another alternative, if the network is small enough, is to look over all the possible configurations of discrete variables. This is mainly a testing method, to ensure that the rest of the approximate algorithms are giving good solutions in small networks. We refer to this alternative as *Exhaustive Search MPE*.

MAP inference entails an even bigger challenge, as the non-queried variables must be marginalised out before maximization. We use an approximate way of computing the probabilities of configurations of the variables of interest and resort to *importance integration* (or *importance summation*) in order to apply the same techniques used for MPE Inference.

Thus, the following four alternatives are implemented and tested in the AMDIST toolbox for MPE and MAP Inference:

- MPE/MAP using Simulated Annealing (local and global) optimization methods.
- MPE/MAP using Hill Climbing (local and global) optimization methods.
- MPE/MAP via sampling.
- Exhaustive Search MPE (not parallel).

In the AMIDST toolbox, the methods described above are implemented by taking full advantage of Java 8 Streams. Since hill climbing and simulated annealing are iterative algorithms, a simple manner of parallelizing the execution is the following: first, draw a certain number of samples from the Bayesian network, according to the probability distribution $\mathbf{x}_1, \mathbf{x}_2 \cdots, \mathbf{x}_M \sim \mathbf{X}$ (all of them satisfying the evidence $\mathbf{X}_E = \mathbf{x}_E$, if there is any); then use each sample as the initial guess for any of the optimization algorithms (hill climbing or simulated annealing), which can be run in parallel to obtain M estimates of the MPE/MAP:

$$\mathbf{x}_j^* = \text{optimizationAlgorithm}(\mathbf{x}_j), \forall j = 1, 2, \cdots, M$$

which are candidates to the optimal solution. Finally, pick the best found solution

$$\mathbf{x}^* = \arg\max p(\mathbf{x}_j^*), 1 \leq j \leq M$$

as the MPE/MAP or, if interested, pick the set of k best-performing solutions, namely those with the greatest probability.

Figure 5.2 shows the stream flow of the Map/Reduce scheme for MPE and MAP Inference. If the parallel mode is activated, then several instantiations of the selected search algorithm are run, based on different starting points. Multiple instantiations of the local variations can also be executed.

The code in Listing 4 shows an example of use of the MPE inference algorithm.

Listing 4: Example of MPE Inference

```
public static void main(String[] args) throws Exception {
```

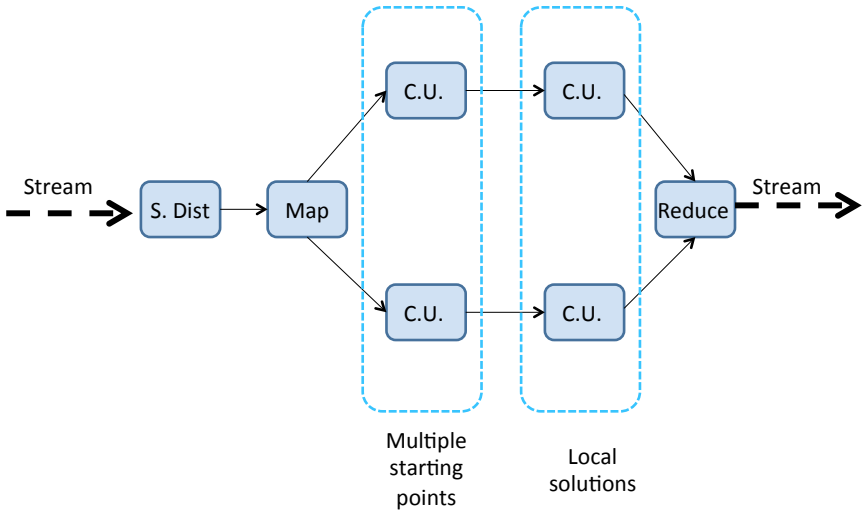


Figure 5.2: Illustration of the Map/Reduce scheme for MPE and MAP Inference.

```
//We load the WasteIncinerator BN, contains multinomial and Gaussian variables.
BayesianNetwork bn = BayesianNetworkLoader.loadFromFile("./networks/
    WasteIncinerator.bn");
List<Variable> topologicalOrder = Utils.getTopologicalOrder(bn.getDAG());

//We recover e.g. two relevant variables: Min (Gaussian) and W (multinomial).
Variable varMin = bn.getVariables().getVariableByName("Min");
Variable varW = bn.getVariables().getVariableByName("W");

//First we create an instance of an MPEInference algorithm.
MPEInference mpeInference = new MPEInference();

//Then, we set the BN model
mpeInference.setModel(bn);

//Set the evidence.
Assignment assignment = new HashMapAssignment(2);
assignment.setValue(varW, 0);
assignment.setValue(varMin, 0.15);
mpeInference.setEvidence(assignment);

System.out.println("Evidence: " + assignment.outputString(topologicalOrder) + "\n
    ");
```

```
        //We can also set to be run in parallel on multicore CPUs
        mpeInference.setParallelMode(true);

        //Then we run inference (uses Hill climbing with local search, by default)
        mpeInference.runInference();

        //We show the found MPE estimate
        System.out.println("MPE = " + mpeInference.getEstimate().outputString(
            topologicalOrder));

        //And its probability
        System.out.println("P(MPE) = " + Math.exp(mpeInference.
            getLogProbabilityOfEstimate()));
    }
```

The `runInference` method can be invoked with an `enum SearchAlgorithm` parameter that represents the search algorithm to use (`EXHAUSTIVE`: exhaustive sequential search; `SAMPLING`: Sampling; `SA_LOCAL`: Simulated annealing, local; `SA_GLOBAL`: Simulated annealing, global; `HC_LOCAL`: Hill climbing, local (by default); and `HC_GLOBAL`: Hill climbing, global).

The code in Listing 5 shows an example of use of the MAP inference algorithm. Similarly to the case of MPE Inference, the `runInference` method can be invoked with an `enum SearchAlgorithm` parameter (this time from the MAPInference class) that represents the search algorithm to use. In this case, the `EXHAUSTIVE` for exhaustive sequential search is not available.

Listing 5: Example of MAP Inference

```
    public static void main(String[] args) throws Exception {

            //We load the WasteIncinerator BN, contains multinomial and Gaussian
                variables.
        BayesianNetwork bn = BayesianNetworkLoader.loadFromFile("./networks/
            WasteIncinerator.bn");
        List<Variable> topologicalOrder = Utils.getTopologicalOrder(bn.getDAG());

        //We recover e.g. two relevant variables: Min (Gaussian) and W (multinomial).
        Variable varMin = bn.getVariables().getVariableByName("Min");
        Variable varW = bn.getVariables().getVariableByName("W");

        //First we create an instance of a MAPInference algorithm.
        MAPInference mapInference = new MAPInference();

        //Then, we set the BN model
        mapInference.setModel(bn);

        //Set the evidence.
        Assignment assignment = new HashMapAssignment(2);
        assignment.setValue(varW, 0);
        assignment.setValue(varMin, 0.15);
```

```
mapInference.setEvidence(assignment);

//Set also the list of variables of interest (or MAP variables).
List<Variable> varsInterest = new ArrayList<>();

Variable var1 = bn.getVariables().getVariableByName("B");
Variable var2 = bn.getVariables().getVariableByName("C");

varsInterest.add(var1);
varsInterest.add(var2);
mapInference.setMAPVariables(varsInterest);

//We can also set to be run in parallel on multicore CPUs
mapInference.setParallelMode(true);

//Then we run inference (Uses simulated annealing with local search)
mapInference.runInference(MAPInference.SearchAlgorithm.SA_LOCAL);

//We show the found MPE estimate
System.out.println("MAP = " + mapInference.getEstimate().outputString(
    topologicalOrder));

//And its probability
System.out.println("P(MAP) = " + Math.exp(mapInference.
    getLogProbabilityOfEstimate()));
}
```

6 Dynamic Inference Engine

The Dynamic Inference Engine module includes all the inference algorithms for dynamic Bayesian networks. It consists of three different methods, namely, Dynamic VMP, Dynamic IS and Dynamic MAP Inference. This module has been designed and implemented to be extendable and support future implementations of other inference algorithms for dynamic models. Please see the appendix for the class diagrams for the dynamic inference packages.

6.1 Dynamic VMP and IS

Inference in dynamic Bayesian networks obviously share the computational difficulties of regular Bayesian networks, but in the dynamic case we are also often faced with additional problems. One is the entanglement problem, where after a certain time step, all variables describing the belief state have become dependent given the observation up to this point, and we can therefore not represent the exact belief state in factorised form. More detailed information can be found in Section 5 of Deliverable 3.1 [4].

In AMIDST we use the Factored Frontier Algorithm [8] to compute the posterior of each

belief state individually. The current implementation includes dynamic versions of the VMP, IS and Hugin algorithms, but it can easily accommodate other static inference techniques.

The code in Listing 6 shows an example of use of the dynamic VMP inference algorithm. The use of Dynamic IS only changes the parameter of the `setInferenceAlgorithmForDBN`. More details about the classes and methods related to this inference method in the AMIDST Toolbox can be found in the appendix.

Listing 6: Example Dynamic VMP Inference

```
//We first generate a dynamic Bayesian network
//(NB structure with class and attributes temporally linked)
DynamicBayesianNetworkGenerator.setNumberOfContinuousVars(2);
DynamicBayesianNetworkGenerator.setNumberOfDiscreteVars(5);
DynamicBayesianNetworkGenerator.setNumberOfStates(3);
DynamicBayesianNetwork extendedDBN = DynamicBayesianNetworkGenerator.
    generateDynamicNaiveBayes( new Random(1),2,true);

//We select the target variable for inference, in this case the class variable
Variable classVar = extendedDBN.getDynamicVariables().getVariableByName("ClassVar
    ");

//We create a dynamic dataset with 3 sequences for prediction.
//The class var is made hidden.
DynamicBayesianNetworkSampler dynamicSampler =
                                new DynamicBayesianNetworkSampler(extendedDBN);
dynamicSampler.setHiddenVar(classVar);
DataStream<DynamicDataInstance> dataPredict =
                                        dynamicSampler.sampleToDataBase(3,100);

//We select VMP with the factored frontier algorithm as the Inference Algorithm
FactoredFrontierForDBN FFalgorithm = new FactoredFrontierForDBN(new VMP());
InferenceEngineForDBN.setInferenceAlgorithmForDBN(FFalgorithm);

UnivariateDistribution posterior = null;
for (DynamicDataInstance instance : dataPredict) {
    //The InferenceEngineForDBN must be reset at the beginning of each Sequence.
    if (instance.getTimeID()==0 && posterior != null) {
        InferenceEngineForDBN.reset();
    }
    //We also set the evidence.
    InferenceEngineForDBN.addDynamicEvidence(instance);

    //Then we run inference
    InferenceEngineForDBN.runInference();

    //Then we query the posterior of the target variable
    posterior = InferenceEngineForDBN.getFilteredPosterior(classVar);

    //We show the output
    System.out.println("P(ClassVar | e) = "+posterior);
}
```

Additionally, the AMIDST toolbox supports exact inference for dynamic Bayesian networks with discrete variables through the HUGIN API. Note that the latter operates on the expanded dynamic Bayesian network and the entanglement problem may still be present.

6.2 Dynamic MAP

The objective of our Dynamic MAP in the AMIDST Toolbox it to establish (variational) posterior distributions over subsets of MAP variables. Once these posterior distributions over non-disjoint subsets have been found, we may define a joint distribution over all the MAP variables based on a suitable combination of the posterior distributions over these subsets.

For a given time step t, we shall denote the observable variables $\boldsymbol{O}_t$, the hidden variables $\boldsymbol{H}_t$, and the control variable Y_t, which is the variable of interest. We assume that a prior Bayesian learning step has been completed, and the posterior (variational) expectations over the model parameters have been used to initialize the model. A straight-forward application of variational mean-field for finding MAP configurations will only provide an approximation of the marginal MAP configuration. We can capture the local dependencies between the MAP variables by aggregating/combining variables that appear together in consecutive time-steps. This partition produces different aggregated models, which we shall refer to as local-dependency models. Figure 6.1 shows the structure of the two only possible local dependency models induced by a pair-wise relations between the MAP variables. Subsequent mean-field inference would then provide posterior distributions over the aggregated variables or, equivalently, the corresponding MAP-subsets.

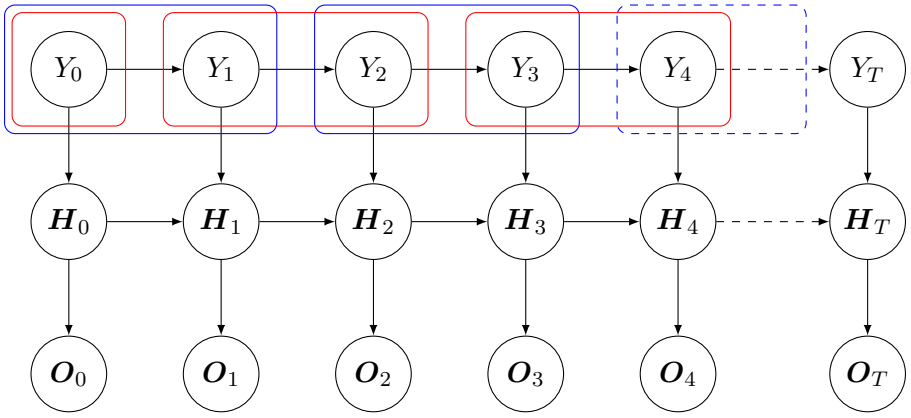
The variational posterior distributions obtained for different local-dependency models can be used to define a posterior joint distribution over $\boldsymbol{Y}_{0:T}$. In AMIDST, we combine the results from, say D, different partitionings as

$$Q^*(Y_0, \ldots, Y_T) = \frac{1}{D} \sum_{i=1}^{D} Q_{P_i}(Z_0, \ldots, Z_T), \tag{6.1}$$

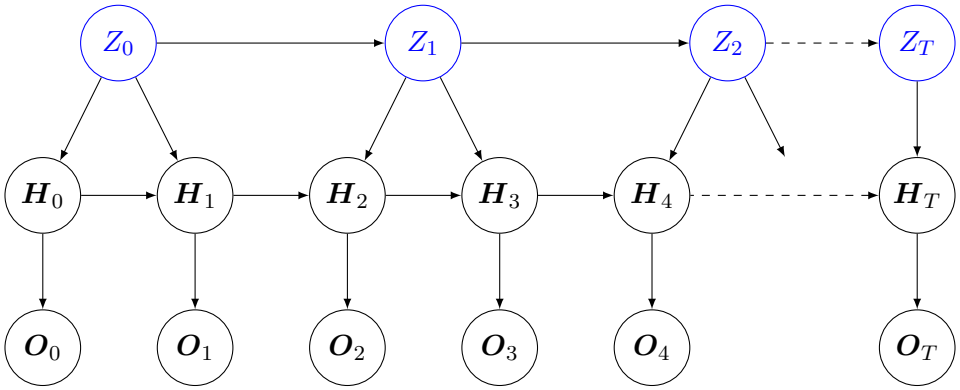
where $Q_{P_i}(Z_0, \ldots, Z_T)$ is the variational posterior distribution captured by partitioning P_i. The probability distributions and density functions of the local-dependency models can be easily calculated based on the original model. Further details have been included in Deliverable 3.4 [14].

An approximate MAP configuration can be found by applying the standard Viterbi algorithm based on the Markov model defined by the distributions above.

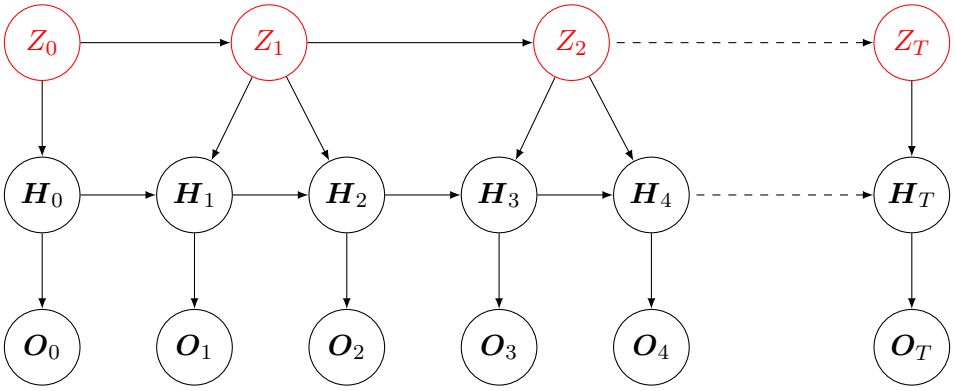
The code in Listing 7 shows an example of use of the dynamic MAP inference algorithm.



(a) Original model. The two only possible partitions for pair-wise relations of MAP variables are marked in red and blue.



(b) Local-dependency model equivalent to the *blue* partitioning.



(c) Local-dependency model equivalent to the *red* partitioning.

Figure 6.1: The figure illustrates the two possible pair-wise partitionings and combinations of the MAP variables in the simplified AMIDST model.

Listing 7: Example Dynamic MAP Inference

```
public static void main(String[] arguments) throws IOException, ClassNotFoundException {

        //We load the dynamic Bayesian network
        DynamicBayesianNetworkGenerator.setNumberOfContinuousVars(3);
        DynamicBayesianNetworkGenerator.setNumberOfDiscreteVars(5);
        DynamicBayesianNetworkGenerator.setNumberOfStates(2);
        DynamicBayesianNetworkGenerator.setNumberOfLinks(5);

        DynamicBayesianNetwork dynamicBayesianNetwork = DynamicBayesianNetworkGenerator.
            generateDynamicNaiveBayes(new Random(0), 2, true);

        //We initialize the dynamic MAP object
        int nTimeSteps = 6;
        DynamicMAPInference dynMAP = new DynamicMAPInference();
        dynMAP.setModel(dynamicBayesianNetwork);
        dynMAP.setNumberOfTimeSteps(nTimeSteps);

        Variable mapVariable = dynamicBayesianNetwork.getDynamicVariables().
            getVariableByName("ClassVar");
        dynMAP.setMAPvariable(mapVariable);

        //We generate evidence for {T=0,...,nTimeSteps-1}
        List<Variable> varsDynamicModel = dynamicBayesianNetwork.getDynamicVariables().
            getListOfDynamicVariables();
        int indexVarEvidence1 = 2;
        int indexVarEvidence2 = 3;
        int indexVarEvidence3 = 4;
        Variable varEvidence1 = varsDynamicModel.get(indexVarEvidence1);
        Variable varEvidence2 = varsDynamicModel.get(indexVarEvidence2);
        Variable varEvidence3 = varsDynamicModel.get(indexVarEvidence3);

        List<Variable> varsEvidence = new ArrayList<>(3);
        varsEvidence.add(0,varEvidence1);
        varsEvidence.add(1,varEvidence2);
        varsEvidence.add(2,varEvidence3);

        double varEvidenceValue;

        Random random = new Random();

        List<DynamicAssignment> evidence = new ArrayList<>(nTimeSteps);

        for (int t = 0; t < nTimeSteps; t++) {
            HashMapDynamicAssignment dynAssignment = new HashMapDynamicAssignment(
                varsEvidence.size());
            for (int i = 0; i < varsEvidence.size(); i++) {
                dynAssignment.setSequenceID(12302253);
                dynAssignment.setTimeID(t);
                Variable varEvidence = varsEvidence.get(i);
                if (varEvidence.isMultinomial()) {
                    varEvidenceValue = random.nextInt(varEvidence1.getNumberOfStates());
                } else {
```

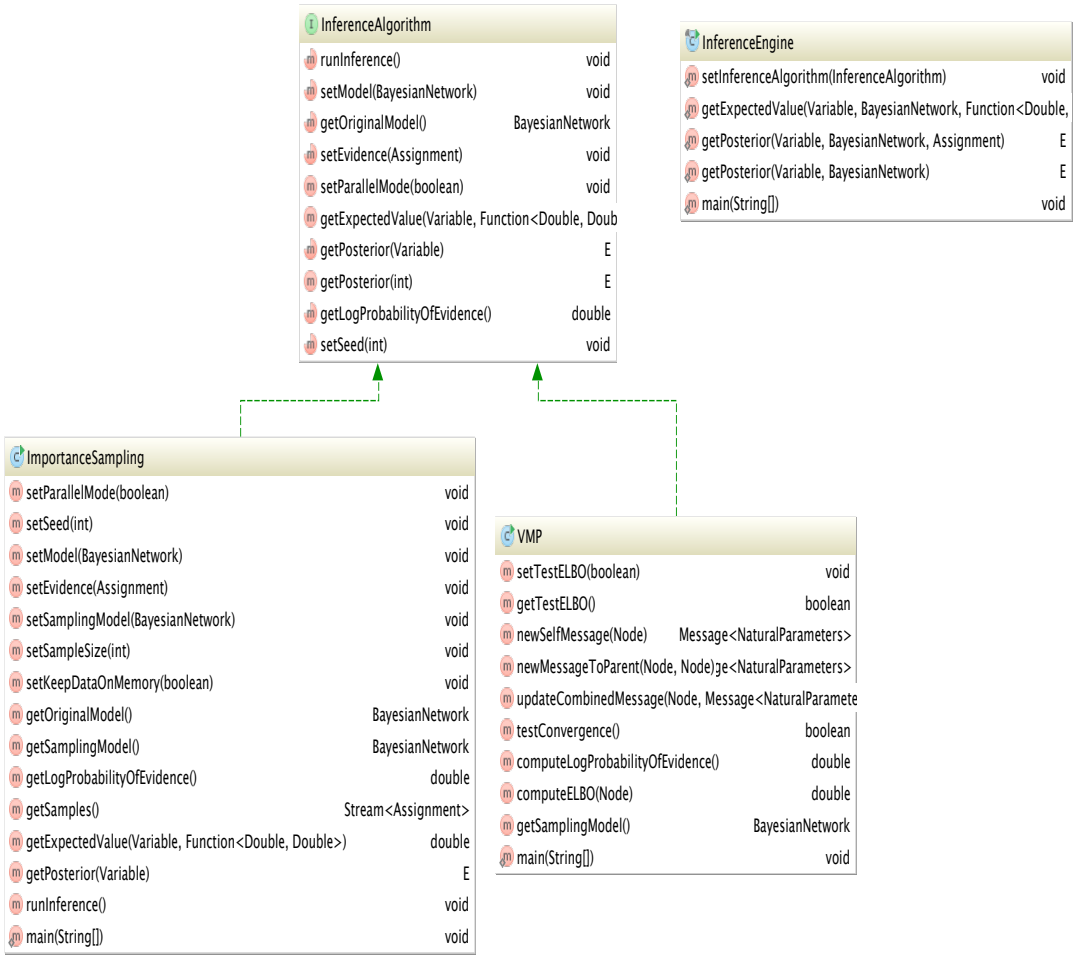
```
                varEvidenceValue = -5 + 10 * random.nextDouble();
            }
            dynAssignment.setValue(varEvidence, varEvidenceValue);
        }
        evidence.add(dynAssignment);
    }

    //We set the evidence and run the inference engine
    dynMAP.setEvidence(evidence);
    dynMAP.runInference();

    //We show the results
    Assignment MAPestimate = dynMAP.getMAPestimate();
    double MAPestimateProbability = dynMAP.getMAPestimateProbability();

    System.out.println("MAP sequence over " + mapVariable.getName() + ":");
    List<Variable> MAPvarReplications = MAPestimate.getVariables().stream().sorted((
        var1,var2) -> (var1.getVarID()>var2.getVarID()? 1 : -1)).collect(Collectors.
        toList());

    StringBuilder sequence = new StringBuilder();
    MAPvarReplications.stream().forEachOrdered(var -> sequence.append(Integer.
        toString((int)MAPestimate.getValue(var)) + ", "));
    System.out.println(sequence.toString());
    System.out.println("with probability prop. to: " + MAPestimateProbability);
}
```

7 Conclusion

This document summarises the prototype software modules integrated into the inference part of the AMIDST toolbox. The Static Inference Engine module includes VMP, IS, exact inference by means of the AMIDST HUGIN API, MAP and MPE inference algorithms. The Dynamic Inference Engine module includes Dynamic MAP and factored-frontier-based versions of the dynamic VMP and dynamic IS algorithms.

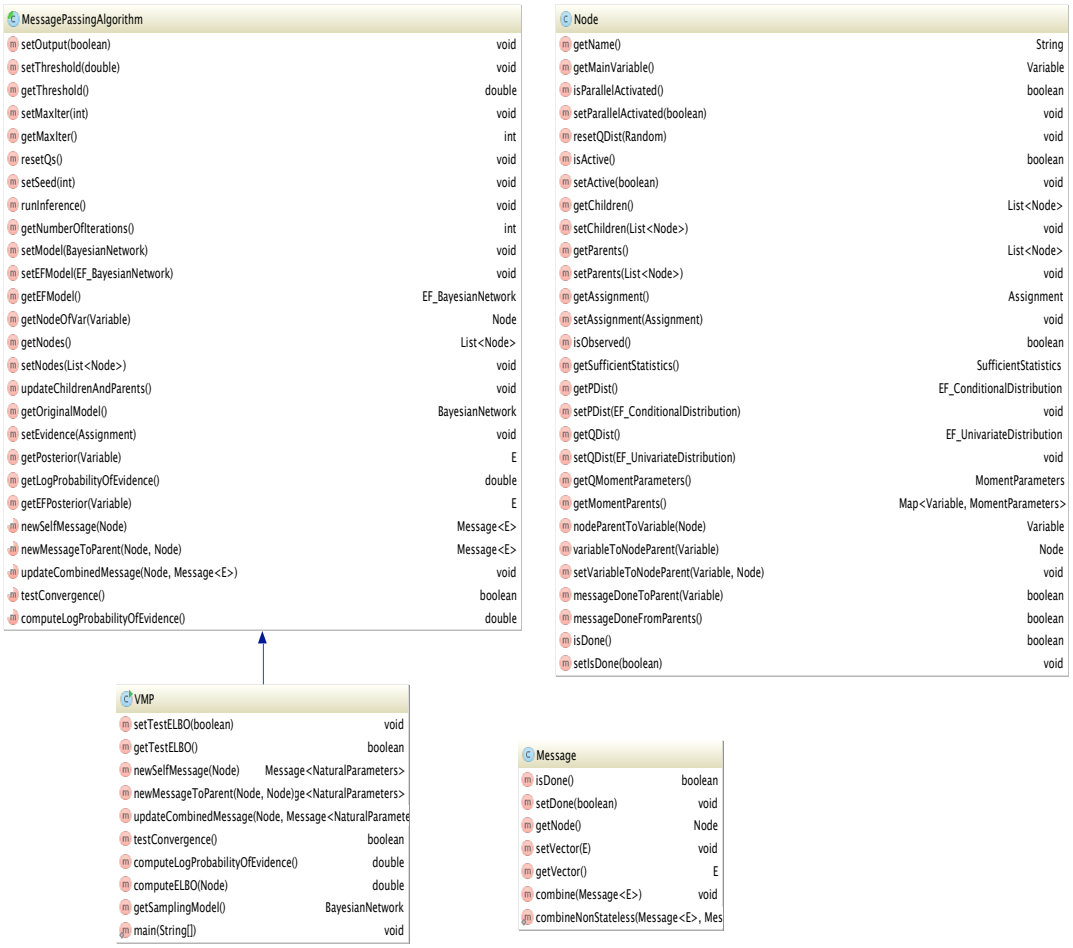
Class diagrams

Package eu.amidst.core.inference

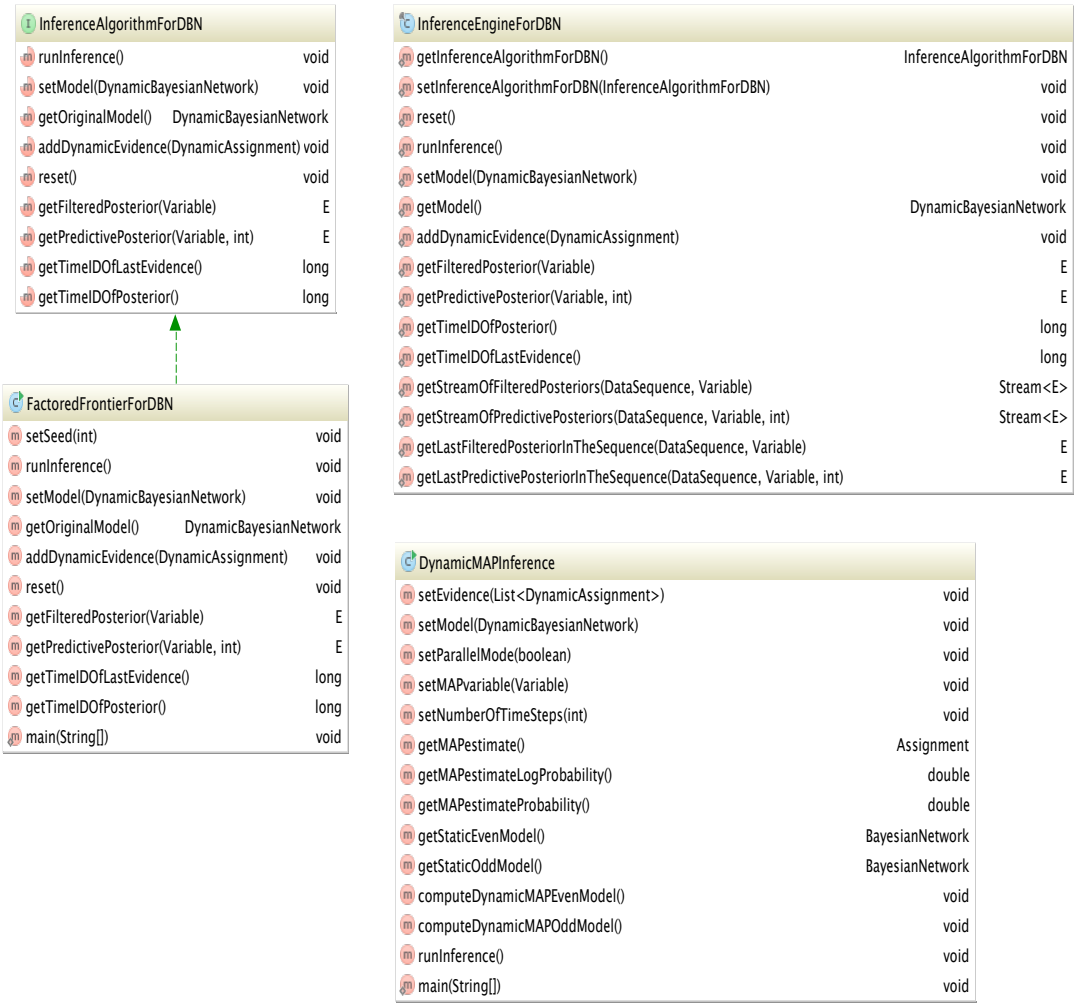


Package eu.amidst.core.inference

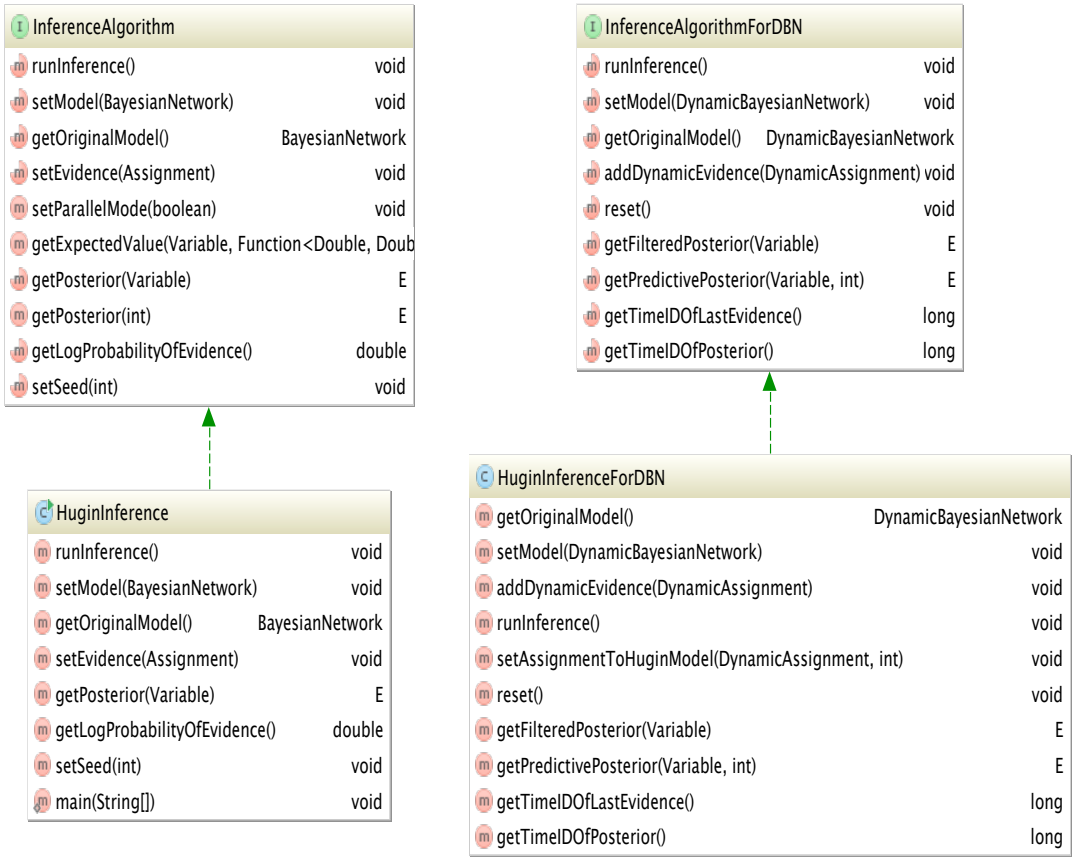
Package eu.amidst.core.inference.messagepassing



Package eu.amidst.dynamic.inference



Package eu.amidst.huginlink.inference



References

[1] Borchani, H., Fernández, A., , Langseth, H., Madsen, A.L., Martínez, A.M., Masegosa, A., Nielsen, T.D., Salmerón, A.: Progress report on software development (March 2015) Deliverable 3.2 of the AMIDST project (FP7 project funded by the EC under contract 616209), `amidst.eu`.

[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 (FP7 project funded by the EC under contract 616209), `amidst.eu`.

[3] 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 – Final report (March 2015) Deliverable 2.2 of the AMIDST project (FP7 project funded by the EC under contract 616209), `amidst.eu`.

[4] Langseth, H., Madsen, A.L., Nielsen, T.D., Rumí, R., Salmerón, A.: State of the art of inference in hybrid and dynamic models (October 2014) Deliverable 3.1 of the AMIDST project (FP7 project funded by the EC under contract 616209), `amidst.eu`.

[5] Winn, J., Bishop, C.: Variational message passing. Journal of Machine Learning Research **6** (2005) 661–694

[6] Yuan, C., Druzdzel, M.: Importance sampling for general hybrid Bayesian networks. In: Proceedings of the 11th International Conference on Artificial Intelligence and Statistics. (2007) 652–659

[7] Gámez, J.A.: Abductive inference in Bayesian networks: A review. In Gámez, J.A., Moral, S., Salmerón, A., eds.: Advances in Bayesian Networks. Volume 146 of Studies in Fuzziness and Soft Computing. Springer Berlin Heidelberg (2004) 101–120

[8] Murphy, K.P., Weiss, Y.: The factored frontier algorithm for approximate inference in DBNs. CoRR **abs/1301.2296** (2013)

[9] Jensen, F.V., Nielsen, T.D.: Bayesian networks and decision graphs. Second edn. Springer Publishing Company, Incorporated (2007)

[10] Pearl, J.: Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)

[11] Kjærulff, U.: A computational scheme for reasoning in dynamic probabilistic networks. In: Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence, San Francisco, California, Morgan Kaufmann Publishers (1992) 121–129

[12] Murphy, K.P.: Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, UC Berkeley, Computer Science Division (2002)

[13] Borchani, H., Fernández, A., Langseth, H., Madsen, A.L., Martínez, A.M., Masegosa, A., Nielsen, T.D., Salmerón, A.: A software library implementation of the modeling framework (March 2015) Deliverable 2.3 of the AMIDST project (FP7 project funded by the EC under contract 616209), `amidst.eu`.

[14] Borchani, H., Ramos-López, D., , Langseth, H., Madsen, A.L., Martínez, A.M., Masegosa, A., Nielsen, T.D., Salmerón, A.: Efficient inference in hybrid domains with static and dynamic models (November 2015) Deliverable 3.4 of the AMIDST project (FP7 project funded by the EC under contract 616209), `amidst.eu`.

[15] Jordan, M., Ghahramani, Z., Jaakkola, T., Lawrence, S.: An introduction to variational methods for graphical models. Machine Learning **37**(2) (1999) 183–233

[16] Attias, H.: A variational Bayesian framework for graphical models. Advances in neural information processing systems (2000) 209—215

[17] Borchani, H., Langseth, H., Nielsen, T.D., Salmerón, A., Madsen, A.L.: Progress report on the software development (December 2014) Deliverable 4.1 of the AMIDST project (FP7 project funded by the EC under contract 616209), `amidst.eu`.

[18] Beal, M.J.: Variational algorithms for approximate Bayesian inference. PhD thesis, Gatsby Computational Neuroscience Unit, University College London (2003)

[19] Hammersley, J., Handscomb, D.: Monte Carlo Methods. Chapman & Hall (1964)

[20] Lauritzen, S.L., Jensen, F.: Stable local computation with conditional Gaussian distributions. Statistics and Computing **11**(2) (2001) 191–203

[21] de Campos, L.M., Gámez, J.A., Moral, S.: Partial abductive inference in Bayesian belief networks by simulated annealing. International Journal of Approximate Reasoning **27**(3) (2001) 263 – 283

[22] Park, J.D., Darwiche, A.: Approximating map using local search. In: Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence. UAI'01, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2001) 403–410

[23] Salmerón, A., Ramos-López, D., Borchani, H., Martínez, A.M., Masegosa, A., Fernández, A., Langseth, H., Madsen, A.L., Nielsen, T.D.: Parallel importance sampling in conditional linear Gaussian networks. In: CAEPIA 2015: Lecture Notes in Artificial Intelligence (LNAI). Volume 9422., Springer (2015) 36–46