# Contents

**1 Executive summary** **3**

**2 Introduction** **4**

**3 Learning Bayesian networks** **5**

**4 TDN – Design** **8**

**5 Task 4.1: Parallelization of structural learning** **8**

**6 Conclusion** **11**

**References** **11**

# Document history

| Version | Date | Author (Unit) | Description |
|---|---|---|---|
| v0.3 | 21/11 2014 | Helge Langseth, Thomas D. Nielsen, Antonio Salmerón | First draft |

# 1 Executive summary

The aim of this document is to describe the progress of the software development related to learning in the AMIDST project at the end of the first year.

# 2 Introduction

**[[ This is just the first page from D3.1, then quickly terminated after learning is introduced. Just to say what a BN is. ]]**

Probabilistic graphical models provide a well-founded and principled approach for performing inference in complex domains endowed with uncertainty. 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 graphical model. Collectively, the two components provide a compact representation of the joint probability distribution over the domain being modelled.

Bayesian networks (BNs) [**?**] are a particular type of probabilistic graphical model that has enjoyed widespread attention in the last two decades. Figure 1 shows a BN representing the joint distribution of variables $X_1, \ldots, X_5$. Attached to each node, there is a conditional probability distribution given its parents in the network, so that the joint distribution factorises as

$$p(X_1, \ldots, X_5) = p(X_1)p(X_2|X_1)p(X_3|X_1)p(X_4|X_2, X_3)p(X_5|X_3).$$

In general, for a BN with $n$ variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, the joint distribution factorises as

$$p(\mathbf{X}) = \prod_{i=1}^{n} p(X_i|\text{pa}\,(X_i)), \tag{1}$$

where $\text{pa}\,(X_i)$ denotes the set of parents of $X_i$ in the network.

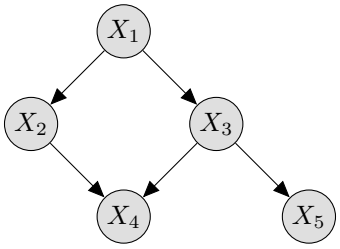


Figure 1: A Bayesian network with five variables.

We will use lowercase letters to refer to values or configurations of values, so that $x$ denotes a value of $X$ and $\mathbf{x}$ is a configuration of the variables in $\mathbf{X}$. Given a set of observed variables $\mathbf{X}_E \subset \mathbf{X}$ and a set of variables of interest $\mathbf{X}_I \subset \mathbf{X} \setminus \mathbf{X}_E$, *probabilistic*

*inference* is the calculation of the posterior distribution $p(x_i|\mathbf{x}_E)$ for each $i \in I$. A thorough introduction to the state of the art for inference in Bayesian networks was given in [**?**].

In this document we assume that inference techniques for a given Bayesian network is available, and will consider how to define a Bayesian network model that fits a data set as well as possible. This process is known as *learning* in the Bayesian network community.

# 3 Learning Bayesian networks

Consider again the factorization of the full joint distribution $p(\mathbf{x})$ in Equation (1). With this factorization we can efficiently represent the joint probability distribution $p(\mathbf{x})$, but it requires that $\text{pa}(X_i)$, i.e., the *parent set* of $X_i$, is known for each $i = 1, \ldots, n$. The parents of $X_i$ are exactly the nodes which have an outgoing edge pointing to $X_i$ (e.g., $\text{pa}(X_4) = \{X_2, X_3\}$ for the model in Figure 1). Algorithms for learning the parent sets, also known as *structural learning* algorithms, follow one of two approaches: *i*) search and score methods, see e.g. [**?**] and *ii*) constraint-based techniques [**?**]. Both will be considered in Task 4.1 of the project, and the progress towards the implementation of structural learning in AMIDST is summarized in Section 5.

As soon as the parent sets in Equation (1) are determined, the specific conditional distributions $p(X_i|\text{pa}(X_i))$ can also be learned from data. The approach taken in the AMIDST project is that the conditional distribution $p(X_i|\text{pa}(X_i))$ is assumed to be a member of the exponential family (including, among others, the Gaussian and multinomial distributions). Therefore, parameter learning amounts to finding the optimal *parameterization* of the given distributions. This task will be considered in Task 4.2 and Task 4.4.

Let $\boldsymbol{\theta}$ denote the combination of all parameters of the model (that is, all parameterizations of the local conditional distribution functions in Equation (1)), and let $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ denote the dataset which we are learning from. $\mathcal{D}$ is a collection of $N$ independent configurations over the variables of the domain. Note that $\mathcal{D}$ may be *incomplete*, in case some of the observations are missing. Missingness can occur, for instance, if some of the variables are latent, or some data-collecting instrument fails.

Now, parameter learning in Bayesian networks amounts to estimating $\boldsymbol{\theta}$ using some optimization criterion that depend on $\mathcal{D}$. The learning generally comes in two different shapes. The most commonly used approach (at least traditionally) is *maximum likelihood* learning, which attempts to find the model parameters that maximizes the *likelihood* of the model given the data. The likelihood of the parameters given the data set $\mathcal{D}$, is defined as $\mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \prod_{j=1}^{N} p(\mathbf{x}_j|\boldsymbol{\theta})$, and the maximum likelihood estimator is $\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}|\mathcal{D})$.

Maximum likelihood learning is very efficient in exponential family models when there are no missing observations in $\mathcal{D}$, because the parameters of each distribution can be

optimized independently. However, closed-form solutions are not available when the dataset contains missing values.

Missing values is, though, a key ingredient in the data sets entertained in the AMIDST project. In these cases, maximum likelihood learning typically relies on the Expectation Maximization (EM) algorithm or generalizations thereof. Implementations of the (generalized) EM algorithm iterates over the following two steps that are repeated until convergence: *i*) E-step: Inference in the model given the data-set (using a model with the current parameterization); *ii*) M-step: Updates of the parameters using the posterior distributions calculated in the E-step. The EM algorithm is a greedy algorithm, that ensures that the likelihood of the current parameter estimates is non decreasing from one iteration to the next, and therefore also converges to a (local) maximum of the likelihood function. Unfortunately, this is only guaranteed when an exact E-step is conducted, and if one uses approximative inference the algorithm may not improve as it moves along. Exact inference in the E-step is computational prohibitive in a streaming context; additionally, the M-step can also be computationally challenging. Maximum likelihood learning is therefore not implemented in the open-source software of the AMIDST project, but can be employed utilizing the interface that the toolbox offers to the implementation inside the Hugin system.

Alternatively to the maximum likelihood approach we have the *Bayesian* paradigm. From a simplistic point of view, the main difference between the two is that the Bayesian set-up regards $\boldsymbol{\theta}$ as a vector of *random variables*, and treats them on an equal footing as all other (unobservable) variables in the domain. From a modelling perspective this extension is straight forward: The model in Figure 2 extends the model in Figure 1 by explicitly representing $\boldsymbol{\theta} = \{\theta_i\}_{i=1}^5$, where $\theta_i$ is the parameterization of the conditional distribution function $p(X_i|\text{pa}\,(X_i))$. Parameter learning now amounts to calculating the probability distribution $p(\boldsymbol{\theta}|\mathcal{D})$, that is, it is reduced to inference in an (extended) Bayesian network model. Thereby, approximate inference techniques like variational Bayes message passing, exponential propagation or importance sampling can be used directly for learning.

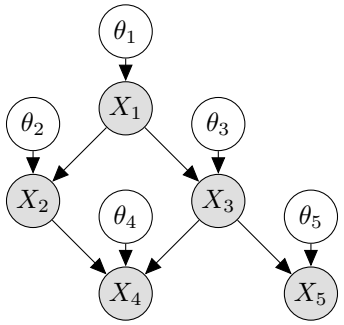


Figure 2: The Bayesian network from Figure 1 extended with explicit representation of the (unknown) parameters.

In practice, the *a priori* marginal distributions for each $\theta_i$ must be declared before the learning can be performed. These prior distributions enables domain experts to express knowledge about the parameterization of the distributions that is combined with information from the dataset to obtain the posterior information captured by $p(\boldsymbol{\theta}|\mathcal{D})$. The prior information consists of the definition of a distributional family for each $\theta_i$ together with a parameterization (the so-called *hyper-parameters*). In AMIDST we will ensure efficient calculation of posteriors by enforcing the distributional families be the conjugate distribution of the likelihood-terms. The hyper-parameters are chosen freely, and this is the vehicle provided to encode prior (expert) knowledge.

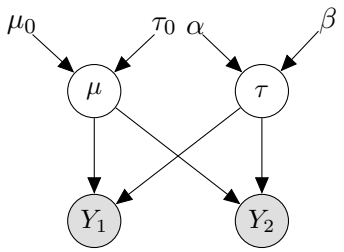


Figure 3: A detailed Bayesian network for the streaming variable $Y_t$ (observed at time $t = 1$ and $t = 2$). $Y_t$ is assumed to follow a Gaussian distribution with mean $\mu$ and precision $\tau$.

Figure 3 shows a detailed description of the model for a streaming variable $Y_t \sim N\left(\mu, \tau^{-1}\right)$ observed at $t = 1$ and $t = 2$. Prior information about these parameters are encoded by assuming $\mu \sim N\left(\mu_0, \tau_0^{-1}\right)$ and $\tau \sim \Gamma(\alpha, \beta)$ for given values of the hyper-parameters $\{\mu_0, \tau_0, \alpha, \beta\}$. The dataset $\mathcal{D} = \{y_1, y_2\}$ enables us to calculate

$$p(\mu, \tau|y_1, y_2, \mu_0, \tau_0, \alpha, \beta) = \frac{p(y_1|\mu, \tau)p(y_2|\mu, \tau)p(\mu|\mu_0, \tau_0)p(\tau|\alpha, \beta)}{p(y_1, y_2|\mu_0, \tau_0, \alpha, \beta)}.$$

The calculation is efficient both in terms of both space and time because the model is from the conjugate exponential family.

It follows that the learning functionality in AMIDST will rest heavily on the implementation of the core components in the toolbox (variables, distributions, Bayesian networks, and so on) as well the design to accommodate efficient inference using these core components. The implementation of inference engines will constitute a key component for the learning implementation because efficient and scalable *inference* is both a requirement and a guarantee for efficient and scalable *parameter learning*. The next section will therefore discuss the top-level design of the AMIDST toolbox, first describing the core components, thereafter moving to the design of the inference components.

# 4 TDN – Design

**[[ This section will cover the design of the SW, as specified through the requirements engineering document. Only the parts related to learning will be covered. ]]**

As indicated in description of work (DOW) document, the objective of the AMIDST project is to provide an open source toolbox implementing novel developments in scalable algorithms for inference and learning with probabilistic graphical models able to operate in hybrid domains.

The initial developed structure of AMIDST toolbox includes eight packages as shown in Figure 4.

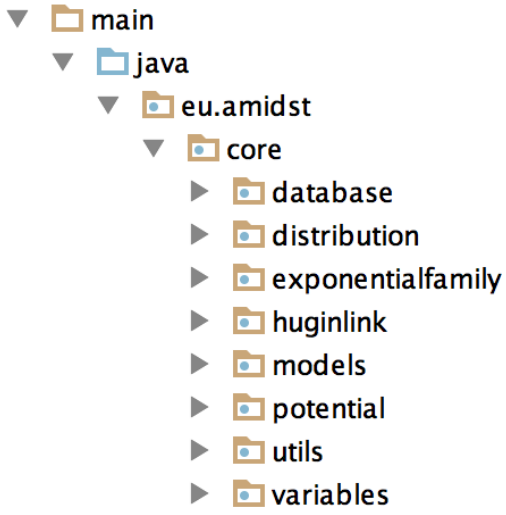


Figure 4: The main packages defined for Amidst toolbox.

# 5 Task 4.1: Parallelization of structural learning

Task 4.1 is devoted to the development of parallel algorithms for structural learning of Bayesian networks. The task covers *structure restricted models* as well as *classic constraint-based methods*.

Structure restricted models are sub-classes of Bayesian networks where only some particular structures are allowed. Typically, such kind of models are employed in specific tasks like *classification and regression* where one is interested in predicting the value of a target variable rather than in accurately modeling the dependencies among the variables in the model. A classification model contains a set of variables $\{X_1, \ldots, X_n, C\}$ where

$C$ is the *class variable* and $X_1, \ldots, X_n$ are called *features*. A Bayesian network can be used for classification purposes by modeling the distribution $p(X_1, \ldots, X_n, C)$. Then, an item with observed feature values $x_1, \ldots, x_n$ is classified as belonging to class $c^*$ given by

$$c^* = \arg\max_{c \in \Omega_C} p(c|x_1, \ldots, x_n) = \frac{p(c, x_1, \ldots, x_n)}{p(x_1, \ldots, x_n)}, \tag{2}$$

where $\Omega_C$ denotes the set of possible classes (i.e. the state space of variable $C$).

By restricting the possible structures for representing $p(X_1, \ldots, X_n, C)$, it is possible to avoid the exponential growth in the number of parameters to learn from data with respect to the number of variables. The extreme case is when all the features are assumed to be independent given the class, which minimizes the number of parameters to learn, as the joint distribution factorizes as

$$p(X_1, \ldots, X_n, C) = p(C) \prod_{i=1}^{n} P(X_i|C).$$

A classifier constructed in this way is called a naive Bayes (NB) classifier, and it corresponds to a Bayesian network structure as depicted in Fig. 5.

A n improvement on the NB is the so-called *Tree Augmented Naive Bayes* (TAN) classifier [1]. In a TAN, the features are arranged as a tree, and all the feature variables have the class as a parent. Figure 6 shows a TAN structure with five feature variables, $X_1, \ldots, X_5$.

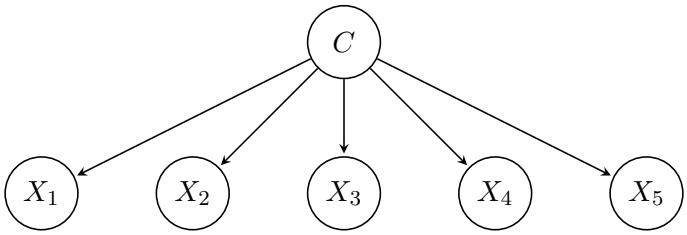


Figure 5: A Naive Bayes structure

The structure of a TAN model is obtained by computing a *score* for each pair of features, namely their *conditional mutual information* given the class. Then, a maximum spanning tree of the features is constructed, labeling the edges using the computed scores.

Structural learning of TAN models has been approached in task 4.1 using two different methods. One of them uses *threads* to distribute the workload onto a number of cores. This method assumes that all data is available in main memory. The algorithm creates a number of threads that iterate through the scores to be computed. Each score is assigned to a thread and a thread only computes the scores assigned to it. The software developed

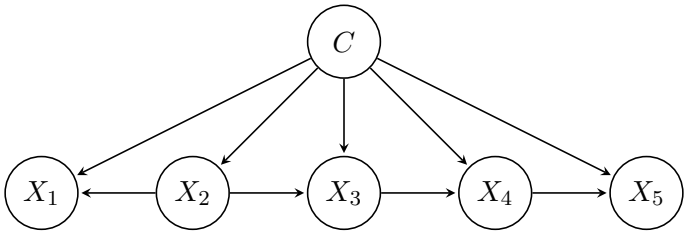


Figure 6: A TAN structure. Unlike in Fig. 5, the features conform a tree rooted at $X_2$

under this approach corresponds to task 5.1 of WP5, and is included in the AMIDST toolbox inside HUGIN AMIDST (see Deliverable D5.1). It can be accessed from the open source AMIDST toolbox using the open source AMIDST $\leftrightarrow$ HUGIN AMIDST interface, called HUGIN-Link, which is a functionality of the open source AMIDST toolbox that enables to use the HUGIN AMIDST API.

The other method uses *MPI* [2] to distribute the workload onto a number of processors. This method does not assume that all data is available in main memory. The variables are distributed between the processors and each process only reads the data for variables assigned to it. The computation of scores is controlled using Balanced-Incomplete Block Designs [3] as described in [4]. This approach assumes that data on each variable is stored in a separate file. This method is not available through the HUGIN-Link interface of the open source AMIDST toolbox. However, it can be accessed from it using file exchange, as the open source AMIDST toolbox is able to read and write HUGIN objects and files, and hence the structure yielded by HUGIN AMIDST can be transferred to software developed using the open source toolbox.

Constraint-based structural learning encompasses those algorithms for inducing unrestricted Bayesian network structures according to the result of a series of conditional independence tests. A brute-force constraint based algorithm could start off with a complete network (where each variable is linked to all the others) and remove edges between variables for which a statistical test accepts the independence hypothesis. Perhaps the most popular constraint-based structural learning algorithm is the so called PC [5]. Basically, the PC algorithm makes use of the same scores as the TAN, i.e. the conditional mutual information scores, which distribution is known to be of class $\chi^2$ under the hypothesis of independence. It allows the construction of a statistical test for deciding about the independence relationships between the variables in the network at any given significance level.

Parallelization of the PC algorithm is still under development and the software will be implemented as part of task 5.1 of WP5. The functionality developed for implementing the parallel learning of TAN models will be used as a part of the parallel PC implementation. It will be available in the AMIDST toolbox as a part of HUGIN AMIDST and its access from the open source AMIDST will be effectively done in a similar way as

described for the parallel TAN.

# 6 Conclusion

This document summarizes the progress of the software development related to learning functionality in the AMIDST toolbox. Parameter learning functionality rests heavily upon efficient design and implementation of core components and inference engines, and these parts were therefore discussed in some detail. Structural learning is ongoing (in Task 4.1) and will be finalized in Month 15. A preliminary discussion of the results was given. Implementation of dedicated functionality for parameter learning (Em algorithm and similar) has not yet started, and is not discussed.

# References

[1] Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Machine Learning **29** (1997) 131–163

[2] Forum, T.: MPI: A Message Passing Interface (1993)

[3] Stinson, D.: Combinatorial designs. Springer (2003)

[4] Madsen, A., Jensen, F., Salmerón, A., Karlsen, M., Langseth, H., Nielsen, T.D.: A new method for vertical parallelisation of tan learning based on balanced incomplete block designs. In: PGM'2014. Lecture Notes in Artificial Intelligence. Volume 8754. (2014) 302–317

[5] Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search. Second edn. MIT Press (2000)