Contents

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 HL – 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.

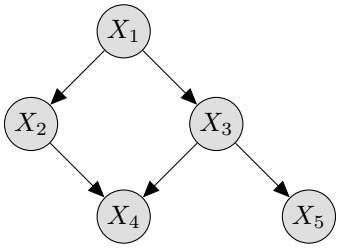

Figure 1: A Bayesian network with five variables.

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 consists of computing 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 denoted *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 belong to a specific distributional family (Gaussian, multinomial, or other member of the exponential family), and learning amounts to finding the optimal *parameterization* of the given distributions. Parameter learning 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) has *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.

Learning parameters when the dataset has missing values, a key ingredient in the data entertained by the AMIDST toolbox, one typically relies on the Expectation Maximization (EM) algorithm or generalizations thereof. Implementations of the (generalized) EM algorithm typically iterates over the following two steps that are repeated until convergence: *i*) E-step: Inference in the model given parameter estimates; *ii*) M-step: Updates of the parameters using inferred states of the variables not observed in the dataset. The EM algorithm is a greedy algorithm that at each iteration guarantees that the likelihood of the present parameter estimates is not lower than the likelihood of the previous estimate as long as the inference algorithm employed by the E-step is exact. Convergence is therefore monitored by keeping track of the likelihood function. If approximate inference is used, no such guarantees exist OR AM I MISTAKEN? NO, apparently they optimize a different function (not likelihood) `http://papers.nips.cc/paper/2404-approximate-expectation-maximization.pdf`.

Alternatively to (generalizations built on) the maximum likelihood approach we have the *Bayesian* paradigm. From a simplistic point of view, the main difference is that the Bayesian set-up regards the parameter-vector $\boldsymbol{\theta}$ as random variables, and treat them on an equal footing as all other (unobservable) variables in the domain. Form 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 θ_i is the parameterization of the conditional distribution function $p(X_i|\text{pa}\,(X_i))$. Learning now amounts to calculating the probability distribution $p(\boldsymbol{\theta}|\mathcal{D})$. Parameter-learning is thus reduced to inference in an (extended) Bayesian network model, and approximate inference techniques like variational Bayes message passing, exponential propagation or importance sampling can be used directly for learning.

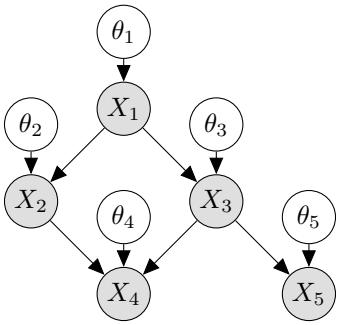

Figure 2: The Bayesian network from Figure 1 extended with explicit representation of the (unknown) parameters.

In practice, the *a priori* marginal distributions for each θ_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 encoded by $p(\boldsymbol{\theta}|\mathcal{D})$. The prior information consists of the definition of a distributional family for each θ_i together with a parameterization (the *hyper-parameters*). In AMIDST we will ensure efficient calculation of posteriors by enforcing the distributional families be the conjugate distribution of the likelihood-terms, leaving the hyper-parameters to be chosen freely to incorporate the a priori knowledge.

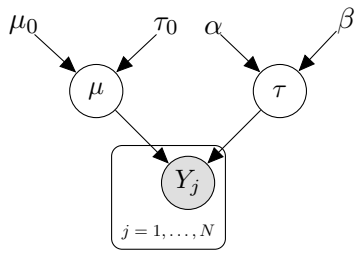

Figure 3: A detailed Bayesian network for the variable Y, which is assumed to follow a Gaussian distribution with mean μ and precision τ. Y is observed N times.

Figure 3 shows a detailed description of the model for a single variable $Y \sim N\left(\mu, \tau^{-1}\right)$. Prior information about the parameters by assuming $\mu \sim N\left(\mu_0, \tau_0^{-1}\right)$ and $\tau \sim \Gamma(\alpha, \beta)$. The dataset $\mathcal{D}$ holds N *i.i.d* observations of Y enabling us to calculate $p(\mu, \tau|\mathcal{D}, \mu_0, \tau_0, \alpha, \beta)$. Since the prior distributions are kept inside the conjugate exponential family, the calculation is efficient both in terms of space and time complexity.

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, etc.) as well the design to accommodate efficient (approximate) 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 and learning 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.]]

5 AS – Task 4.1: Parallelization of structural learning

This section will give updates from Task 4.1, with focus on the actual tool-box implementation of parallel PC. We have two sources of information/ideas how to proceed:

- The slideset/poster we discussed some time ago.
- Hugin people are investigating lines for parallelizing the PC using multi-thread and relying on the TAN-PGM paper.

We should also discuss the learning of TAN classifiers (the PGM paper). Anders has suggested to Antonio that we could include the PGM paper here (except maybe the experiments, that could be reported in D5.1). In this way there would be no rush for having it implemented in the toolbox.

The setup of this section is that we show in practice how the initial procedures described in the design section is to be used.

6 HL – Initial results from the other tasks

[[This section will discuss any results or ideas that have been developed in Tasks 4.2, 4.3, 4.4. Not much to include. Maybe delete?]]

7 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.