Università degli Studi di Milano Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Informatica

LEARNING OF BAYESIAN NETWORKS WITH MISSING DATA

Relatore: Prof. Fabio STELLA

Co-relatore: Ph.D. Alessandro BREGOLI

Tesi di Laurea Magistrale di:

Andrea Ruggieri

Matricola 806808

Anno Accademico 2019-2020

You never fail until you stop trying.
Albert Einstein

Acknowledgements

Special thanks to my Teacher *Fabio Stella* for following me for the duration of the stage. His availability was fundamental for the realization of this project and our collaboration proved to be an important and unforgettable moment of growth for me.

Many thanks to Ph.D. *Marco Scutari* from the Swiss AI Lab IDSIA of Lugano and founder of the `BNlearn` library. He was an active participant in all the presentations and always provided us with very useful advice.

Many thanks to *Francesco Stranieri*, my adventure companion. We had been working on different projects and I feel lucky to have met someone like him.

Many thanks to my correlator *Alessandro Bregoli* and all the *MADLAB* team who followed our presentations remotely every months (I hope they were not bored).

A warm embrance to my special girlfriend *Paola*.

Many thanks to my *Family*, my sister *Daniela* and all my close friends and, in particular, *Andrea* and *Alberto*.

Summary

In the literature, different approaches to replace the missing data are available. These algorithms play an essential role in data mining because the presence of the missing values can lead to the absence of important information by introducing bias into the data and by causing problems such as loss of efficiency when we need to train machine learning models. Therefore, in order to minimize these problems, it is important to study these approaches carefully with the goal of replacing the missing values in the most correct way.
In this thesis, we are going to implement and analyze the *Expectation-Maximisation algorithm*, a recent likelihood-based approach, in three different versions in order to extract the most important properties. Through a complete discussion, we are going to be able to report useful insights and to advice the reader about which EM approach needs to be applied when working with partial datasets.
However, to the best of my knowledge no public implementations nor results are available on the Expectation-Maximisation algorithm and this represents a very stimulating challenge.

Contents

List of Symbols

θ	Parameters of a Bayesian Network	2.4
$\hat{\theta_0}$	Initial values of the parameters	2.4
ϵ	Threshold values used for the stopping criteria	2.4
X	Query variable	2.4
$o[m]$	Partially specified data case in the m-th position of the dataset	2.4
$H[m]$	Variables whose values are missing in the data instance $o[m]$	2.4
$h[m]$	A single assignment for the set $H[m]$	2.4
$\xi[m]$	m'th training example	2.4
U	Set of parents for the query variable X	2.4
$Q(X,U)$	Set of all marginal probabilities	2.4
$\bar{M_\theta}[u]$	Expected sufficient statistics	2.4
$< var_1^1, var_2^2 >$	Instance of a data	2.4
$M[x^0, y^0]$	Number of times in which the instance $< x^0, y^0 >$ appears	2.4
I	Indicator function	2.6
γ	Hidden (unobserved) variables	3.1
$Prop$	Percentile of missing values into a dataset D	4.1
$p(x)$	Probability density vector	4.4
$KL[p(y)\|\|p(x)]$	Kullback-Leibler Divergence given two probability density vectors	4.4
GT	The actual parameters of the Baysian Network. It is the Ground truth	4.4

List of Figures

List of Tables

Listings

Chapter 1

Introduction

When we deal with data, a large number of problems that need to be solved before implementing a machine learning task may arise. In particular, in a machine learning process, we can distinguish different steps. The first two are: the **collection step** and the **preprocessing step**. Typically, the quality of the first step may significantly affect the second and the next steps. Normally, when we deal with low data quality, even the results are of low quality. In a real world this is a very relevant issue because in most cases, data might be **incomplete**, **missing**, **noisy** (e.g. errors or outliers) or **inconsistent**. Therefore, we need data that allow us to make decisions with a good level of reliability [16]. The preprocessing step provides us with different tools to achieve this goal. In this work we are going to focus our attention on **data cleaning step** which includes all activities which are necessary to replace missing values.
In this thesis we are going to discuss carefully the problem associated with missing data. This is a very relevant issue because the presence of missing values might raise different problems as **loss of efficiency**; less patterns extracted from data or conclusions statistically less strong; **complications** in handling and predicting the missing data and **bias** resulting from differences between missing and complete data. In addition, different machine learning models do not work very well in presence of missing values and sometimes the results obtained by the training of a machine learning model are very poor *(garbage in, garbage out)*. To conclude, it is necessary to note that the presence of missing values implies an **absence of information**. Sometimes these information are fundamental because the absence of data can make the problem impossible to solve or a strategic decision difficult to make. Therefore, solving the problem of missing values becomes fundamental in many contexts.
We are going to exploit the basics of *structured learning* and in particular, the *Bayesian Networks*, in order to learn the distribution of complete data as to replace the missing values. This is the fundamental of *Expectation-Maximisation algorithm*.
In Chapter 2 we are going into detail about the issue discussed above, also by providing some theoretical notions. In Chapter 3, we are going to present the methodological approach adopted for the implementation of the project. In the Chapter 4 we are going to deal with the experiments carried out and we are going to describe the most relevant results. In this part, we are going to present the datasets we have selected and we are going to describe the evaluation metrics and the approach adopted for the generation of

missing values. Chapters 5 and 6 concern the conclusions of the work carried out and future developments. Eventually, the results of each experiment are going to be available in the appendix section.

1.1 State of the art

A high number of authors have approached the problem of replacement of missing values using *EM (Expectation-Maximisation) algorithm*. Over the years different papers and books were published and this algorithm seems to have many possible applications. In 1998 ***Friedman*** published *The Bayesian Structural EM Algorithm* [11] and this article remained a reference point in this research area. In particular, the initial problem that Friedam raised concerns how to learn the structure of a Bayesian Network with missing data or hidden variables. In spite of the great success of Expectation Maximisation algorithm, the pioneers have always limited their attention on the hard version of EM algorithm (we call this version *EM Hard*) because it results faster and simpler than the *complete version* (called *EM Soft*). Theoretically, Friedman presents *EM Soft* algorithm [15], by showing several examples and some theoretical results. Anyway, no one has ever tried to implement this version and no results are available.
The study of the EM algorithm is very relevant for imputating the missing values. Authors have proved the important role of the EM algorithm that is one of the best algorithm in the literature. The pioneers agree that EM imputations are better than *mean imputation* and others techniques because they preserve the relation with other variables, which is vital if you use something like *Factor Analysis* or *Linear Regression* [9]. Moreover, this thesis tries to understand if the version EM Soft could lead to improvements at a practical level because nobody has ever tried to provide a complete treatment of this version.

Limits

At the implementation level, one of the most relevant problem concerns the use of exact inference into the expectation step. It may be a problem when the number of missing data is too high because it is **computational inefficient**: the algorithm may require millions of iterations to solve a more complex problem. In this way spatial and temporal complexity increases exponentially with respect to the number of variables. A second problem of the exact inference is the **inefficacy**: the choice of this type of inference is not recommended when it is necessary to make decisions quickly (e.g. automatic driving of a vehicle) and getting the solution too late could be completely useless. In the literature, all the most important implementations of the EM algorithms are based on approximate inference [25]. However, the use of this type of inference could lead to worse results.
Others limits are represented by the poor results and documentation in the literature concerning the comparison between EM Hard and EM Soft. Most of the papers refer purely to the hard version because it is easier than the soft version. In the previous section 1.1 we have mentioned that some theoretical results have been obtained when we compare the two versions of EM algorithm. These results, however, are only partial because they don't take into account the properties of the initial dataset; the structure of the network that we are considering; the number, the type and the distribution of missing values and the

probability distribution of the variables. From these partial observations EM hard would be the preferred version. In other words results using the soft version of EM algorithm are very poor.
Looking at the others imputation methods [13], EM results more difficult to understand and to implement than many other techniques. Researchers have tried, for several years, to evaluate if this imputation technique offers relevant benefits in comparison to other algorithms. As mentioned in the previous section 1.1, EM is able to guarantee better accuracy and precision than the simplest algorithm as the mean replacement. However, different other sophisticated imputation methods have been taken into account. For example, Imputation using *Multiple Imputation*, in some cases seems to be more viable as compared to Expectation Maximization, since the distribution of the data after imputation remained the same. The researchers have been recommended that the type of missingness present in the data, the properties of the data could affect the choice of imputation algorithm because selecting the best algorithm was an extremely difficult task.
Despite these considerations, Expectation Maximisation algorithm remains a reference point in the area of imputation techniques and in the next chapters we are going into the details of this algorithm.

1.2 Goals of the project

As mentioned before, the final goals of this project are the following:

- Planning and implementing a package that makes available the two versions of the EM algorithm: Hard EM and Soft EM.
- Evaluating and experiment the two versions by varying the complexity of the datasets and adopting different strategies for the generation of missing values.
- Providing useful tips for the people who have to deal with missing datasets and need to train a BN model through observation of the results that have been extracted.

More specifically, we are going to ask if the two versions of the EM algorithm exhibit significant differences depending on a number of factors as: the type of missingness, probability distribution of variables, dataset balancing, size of the dataset etc.
By reading this thesis the reader, when will have to deal with missing values, will be able to easily apply the most suitable version of the EM algorithm, with the final goal of replacing them efficiently. In particular, through a full treatment of this argument, the reader will be able to understand the difficulties that may arise when we deal with missing data and he will expand his knowledge.

Chapter 2

Bayesian networks, Missignness and EM algorithm

In this chapter we provide some theoretical notions that are extremely important to clarify the context of this thesis. More precisely, after providing some relevant notions concerning the Bayesian networks, we are going into detail of the missigness problem and we are going to present the EM algorithm: one of the most important ways to replace the missing values.

2.1 Introduction to Bayesian networks

Bayesian networks are a way to represent uncertainty that is consistent with the axioms of probability theory. Bayesian networks are directed, acyclic graphs that allow us to encode the cause effect and conditional independence relationships among variables in the probabilistic reasoning system, where nodes of the graphs are the variables of the domain one wants to model [6]. However, it is important to remark that Bayesian networks are different from *causal networks*. In fact, in BNs arcs do not necessarily imply causation. On the contrary, when we deal with the casual networks, arcs always imply causation.
It is possible to estimate the structure (relationships among variables) through a wide variety of algorithms present in the literature. However, in this thesis we suppose that the structure of the net is well-known. This is the **qualitative component** of a Bayesian network. This part includes the construction of the network by identifying the relationships among the nodes (or variables). When we deal with Bayesian networks we have a second component to take into account. This is the **quantitative component**. All variables have associated a conditional probability table (CPT) which describes the conditional probability of a variable X given the state of the its parents $P(X|parents)$. More precisely, we can define a Bayesian network by providing the following formal definition:

Bayesian networks (BNs) are a class of graphical models, defined by:

- a *network structure*, more precisely a directed acyclic graph (DAG) $\mathcal{G} = (V, A)$, in which each node $v_i \in V$ corresponds to a random variable X_i describing some quantities of interest;

- a *global probability distribution* $\mathbf{X}$ with parameters Θ, which can be factorised into smaller local probability distributions, according to the arcs $a_{i,j} \in A$ present in the graph.

The main role of the network structure is to express the conditional independence relationships among the variables in the model through graphical separation. As a result, $\mathcal{G}$ induces the factorisation:

$$\mathrm{P}(\mathbf{X} \mid \mathcal{G}, \Theta) = \prod_{i=1}^{N} \mathrm{P}\left(X_i \mid \Pi_{X_i}, \Theta_{X_i}\right), \tag{2.1}$$

Usually, in our experiments, the quantitative component must be learned by the EM algorithm through the *Expectation step* (or *E-Step*). Initially, we suppose that the initial probability distribution is unknown and we will set to the uniform distribution for all variables of the network. As we shall see, the algorithm will already be able to learn the correct probability distribution since the first iterations. After learning this probability, the algorithm will be able to estimate the value of the missing values present into the initial dataset. This operation is possible through the *Maximisation step* (or *M-Step*).
Before proceeding with a more detailed description of the EM algorithm it is very important to remark that it is possible to apply the *Structural EM algorithm* to learn an accurate network structure also in presence of missing data. In fact, the EM algorithm is created with the goal of fulfilling this task. Although learning the structure of a Bayesian network is a relevant task, in this thesis, our attention will be focused exclusively on the replacement of missing values. For this reason, we fix the network structure a priori.
In addition, it is important to remark that in terms of distributional assumptions the literature has mostly focused on three cases: Discrete BNs (DBNs), Gaussian BNs (GBNs) and conditional linear Gaussian BNs (CLGBNs). In this project we focus only on DBNs, in which both X and the $X_i|\prod_{x_i}$ are multinomial.
This consideration concludes a first smattering of the EM algorithm, in the following sections we go into detail about the issue that has arisen.

2.2 Missingness problem

In the Chapter 1 we have mentioned the main problems that can arise when we work with missing values. However, it was a simple introduction. In fact, when we have missing values in a dataset, it is very difficult distinguish what kind of missing data we are looking at. For example, some types of missing values are ignorable while other types are non-ignorable and a full treatment has to be done in order to replace the missing values. We call **partially observed variable** a variable of which we observe only a few samples. For example, if we have 100 cases, a variable X is partially observed if we count 90 observed samples and 10 not-observed samples. If a dataset D consists of partially observed variables, then we call it **partial dataset**. On the contrary, a dataset is complete if each instance assigns a value to all the variables in our domain. In a real case, after fulfilling the ingestion step, it is very difficult to work directly with complete data because in some cases, data are missing by accident. There are different reasons that lead to the presence of missing data for example, some fields in a questionnaire may have been omitted by the user during the

compilation phase. In other cases, observations (a set of data) were simply not made; in a medical-diagnosis setting, for example, one never performs all possible tests or asks all of the possible questions. Eventually, some variables are *hidden*, in that their values are never observed. For example, some diseases are not observed directly, but only via their symptoms [15].
We call *incomplete data* an observation which consist of at least one missing value. For example, a user may not fill in all the fields of a form as the annual income.
Generally, the presence of a partially observed variable into a Bayesian network may cause several problems. For example, if we know the data but we do not know the BN structure, how can we learn the structure of the BN in presence of missing value? In addition, Given a DAG (direct acycling graph), how do we estimate the local parameters distribution of a variable containing missing observations? In this case, a bias could be introduced into the network. In particular, the distribution learned from the missing values might deviate from the actual probability distribution and this difference could propagate within the network by making wrong inferences.
One of the most important issue when we work with missing data is to determine the types of missing data. Missingness is broadly categorized in three different categories:

- **Missing completely at random (MCAR):** there is no relationship between whether a data point is missing and any values in the data set, missing or observed. The missing data are just a random subset of the data.
  Generally, data are regarded as being MCAR when data are missing by design, because of an equipment failure or because the samples are lost in transit or technically unsatisfactory. The statistical advantage of data that are MCAR is that the analysis remains unbiased [23]. For example, a questionnaire might be lost in the post, or a blood sample might be damaged in the lab. All these case generate a MCAR data.

- **Missing at random (MAR)**: the propensity for a data point to be missing is not related to the missing data, but it is related to some of the observed data. In this case the missingness is conditional on another variable.
  It is not specifically related to the missing information. For example, if a child does not attend an educational assessment because the child is (genuinely) ill, this might be predictable from other data we have about the child's health, but it would not be related to what we would have measured had the child not been ill. Some may think that MAR does not present a problem. However, MAR does not mean that the missing data can be ignored.

- **Missing not at random (MNAR)**: there is a relationship between the propensity of a value to be missing and its value.
  For example a person does not attend a drug test because the person took drugs the night before [3]. The cases of MNAR data are problematic. The only way to obtain an unbiased estimate of the parameters in such a case is to model the missing data but that requires proper understanding and domain knowledge of the missing variable.

[2]. It is crucial to note that the main imputation algorithms including the EM algorithm work on all types of missing data just presented. Anyway, the performance of each imputation method could strongly depend on the type of missingness. However, the first two types (MCAR and MAR) are both considered **ignorable**. On the contrary, data of type

MNAR are **non-ignorable** since the missing data mechanism itself must be modelled.
We now return to examine the context of the Bayesian networks. It is possible to observe that if all data are complete, then each variable will have associated a local distribution $X_i \sim P(X_i|\Pi_{Xi})$. We remember that the $i-th$ variable is complete if it does not present missing values for all instances. If we are in the opposite condition (partial observed variables), we will have that the i-th variable X_i presents at least a missing values. In this case, estimating the local distribution for the variable X_i is more difficult than the previous case and it is necessary to keep separate the discussion concerning the MAR and MCAR cases from the MNAR case. In the MAR and MCAR cases we have:

$$X_i \sim \begin{cases} P(X_i|\Pi_{Xi}) \; for \; the \; data \; observed \; X_i^{(O)} \\ P(X_i|\Pi_{Xi}) \; for \; the \; missing \; data \; X_i^{(M)} \end{cases} \tag{2.2}$$

vice versa, in the case of MNAR:

$$X_i \sim \begin{cases} P(X_i^{(O)}|\Pi_{Xi}) \; for \; the \; data \; observed \; X_i^{(O)} \\ P(X_i^{(M)}|\Pi_{Xi}) \; for \; the \; missing \; data \; X_i^{(M)} \end{cases} \tag{2.3}$$

Where M is the **missing mechanism**. More precisely, M is non-ignorable since it cannot be properly estimated by local distribution.
It is important to highlight that there are a number of hypothesis tests that can be carried out to test for MCAR. For example, *Little* [19] developed a test based on means under the different missing data patterns. *Listing and Schlittgen* proposed a simple test based on means [18] and secondly a non-parametric procedure which combines several Wilcoxon rank sum tests [17]. These tests are very relevant because they provide a global view of the missingness mechanism. In the Little Test, the null hypothesis is that the data are MCAR. If the data are not MCAR, the mean scores at each assessment will vary across the patterns. Another important test is the *Fairclough logistic regression* which was used to determine whether the current score was a predictor of reminder-response, suggesting MNAR [10].

2.3 Imputation methods

The presence of missing data represents a very crucial problem. In the literature, statisticians have dealt carefully this issue and over the years, they have presented different methods that can be used to handle missing data. Generally, it is a good rule to split these methods into five broad categories:

- Methods that ignore missing observations;
- Single imputation methods;
- Other imputation methods;
- Likelihood-based methods;
- Indicator methods.

These methods present advantages and disadvantages the we are going to present [2].

Methods that ignore missing observations

This category includes ***Complete case analysis*** and ***Available case analysis*** method. The complete case analysis is the most popular and well-known method in this class. In complete case analysis, all the observations with incomplete data are removed from the analysis.
The disadvantages of each method presents in this category is that if the data are MCAR the reduced dataset would represent a randomly drawn sub-sample of the original data and thus inferences made from this approach are valid. However, in practice the data are not MCAR and participants with complete data may be a biased sub-sample of all participants. As a consequence, a complete case analysis would usually produce biased results [2].

Single imputation methods

these methods attempt to estimate the values of the missing data and 'fill-in' or impute new values. A dataset is considered *complete* when all missing values have been replaced with values, so the dataset has not missing values.
This area includes ***Last value carried forward***, ***Mean substitution***, ***Imputation Using k-NN*** and ***Regression method***.
The disadvantages of all the single imputation methods listed above are that they impute the same missing value every time. In a structured learning study this could lead to biased results. In addition, these methods under-estimate the variance. In addition, it is important to remark that K-NN is quite sensitive to outliers in the data.

Other imputation methods

As single imputation methods lead to an under-estimate of the variability in the dataset, other methods have been developed to combat this problem.
This area includes ***Multiple imputation***,
textittextbfMultivariate Imputation by Chained Equation (MICE) and ***Markov-chain imputation***.
In this class of imputation methods the Multiple imputation is the most popular and well-known method. In particular, Multiple imputation replaces each missing value by M possible values to create M complete datasets. Typically M is between 5 and 10. Single imputation does not take into account the uncertainty in the imputations. After imputation, the data is treated as if they were the actual real values in single imputation [12].

Likelihood-based methods

These methods are more robust than the imputation methods described as they have good statistical properties.
This category includes the ***Expectation-Maximisation algorithm***. We are going to analyze very carefully this algorithm since the next section.

Indicator methods

This final approach to handling missing data attempts to incorporate the missing data patterns into the data analysis.

This area includes ***Indicator method*** and ***Pattern-mixture models***.

Method	Bias on MCAR	Bias on MAR	Bias on MNAR	Handling of variability
Complete cases	NO	highly biased	highly biased	Under-estimates variability in the dataset
Mean imputation	NO	highly biased	highly biased	Under-estimates variability in the dataset It is better than Complete cases
Regression methods	NO	NO	highly biased	Under-estimates variability in the dataset It is better than the previous two
Multiple imputation	NO	NO	highly biased	Produces good estimates of variability in the dataset
Markov-Chain imputation	NO	NO	NO	Produces good estimates of variability in the dataset
E-M Algorithm	NO	NO	NO or low	Produces good estimates of variability in the dataset
Indicator method imputation	NO	NO	highly biased	Under-estimates variability in the dataset
Pattern mixture models	NO	NO	NO	Produces good estimates of variability in the dataset

Table 2.1. Comparison among the main imputation methods

In conclusion, there is no perfect way to compensate for the missing values in a dataset. Each strategy can perform better for certain datasets and missing data types but may perform much worse on other types of datasets. There are some set rules to decide which strategy to use for particular types of missing values, but beyond that, it is necessary to experiment and check which model works best for your dataset [1].

2.4 The Expectation-Maximisation algorithm

The EM algorithm is an iterative procedure, which aims to estimate the missing values and it consists of two steps in each iteration: the Expectation step (E-step) and the Maximisation step (M-step). The Expectation-Maximization algorithm is logically defined as follows:

```
1 Begin Procedure EM_algorithm(...)
2     ∀θ ∈ BN set an initial value θ̂_0
3     while (|θ̂_{j-1} − θ_j| < ε and !Max_iterations) repeat:
4         E-Step: computing the posterior probabilities using exact
              inference
5         M-Step: compute new θ̂_j given P(X_i^(M) | X_i^(O), θ̂_j)
6     θ = θ̂_j
7 End Procedure
```

Listing 2.1. EM procedure using pseudo-code

We examine the procedure just presented. In the second line we have initialized all parameters. We can adopt different strategies for this step. For example, in the the experiments we are going to set an uniform probability distribution for each variable. However, it is possible to adopt other types of strategies as a random initialization. An important advice is to avoid to initialize the probabilities with values closely to 1 or 0. This may not guarantee a fast convergence of the algorithm.
EM is an iterative algorithm, for each iteration, EM computes the expectation step and the maximisation step sequentially. It is very important to set a stopping criteria (line 3) because the algorithm is executed indefinitely (as the *gradient descent algorithm*). At the end of a iteration, the difference between the new parameters and parameters calculated on the previous iteration is computed. If this difference results minor than a threshold

ε then, the stopping condition will be satisfy and the algorithm stops its execution. On the contrary, if this difference results equal or greater than the threshold ε then, the algorithm will continue its computation with an other iteration. It is important to remark that ε is a hyper-parameter and the value can be changed by the user before executing the EM algorithm. A high value of ε guarantee few iterations but it causes a relevant loss of precision. On the contrary, a too low value implies that a great (and useless) number of iterations are necessary. In addition to ϵ, EM allows to set another hyper-parameter called `Num_iterations`. This hyper-parameter defines the maximum number of iterations that EM can execute independently of the difference defined by the threshold ϵ. Anyway, we are going to provide further details in the Chapter 3; in this moment it is very important to highlight that the setting of these hyper-parameters is very crucial when we execute the EM algorithm and the choice should take several factors into account as: the total number of missing values, time requirements, the dimensionality of the dataset etc.
Lines 4 and 5 represent the core of the algorithm. Expectation step imputes the missing data with their posterior expectations or their maximum likelihood estimated using the current BN. On the contrary, Maximisation step uses the complete data with the goal of learning the new parameters θ_j. In fact, the main goal of the E-step is to resolve an inference problem where a missing value represents an unobserved node. We use the exact inference because the approximate inference could lead to poor quality results. In addition, the approximate inference may not guarantee the convergence of the EM algorithm. We are going to highlight this very relevant property in Section 2.5. So, let us use the theory of the inference by enumeration. The aim is to calculate:

$$P(X|e) = \alpha P(X, e) = \alpha \sum_{\gamma} P(X, e, \gamma) \tag{2.4}$$

Where X is our query variable that correspond to a not observed variable, e represents the set of the observed variables, while γ represents the set of the hidden variables. α is a normalisation factor. A summation must be defined for all hidden variables. For this reason the inference by enumeration could be very expensive, especially when the presence of observed variable is rare. The maximisation step is almost linear, it exploits the complete data (derived after the imputation of the missing data) and it iterates on the entire dataset and it computes the new parameters.
Line 6 represents the end of the EM algorithm. In this statement, the final parameters θ match with the parameters computed by the EM algorithm in the last iteration $\hat{\theta}_j$. This conclude the first approach with the EM algorithm. Now, we go into details about the Expectation step.

The Expectation Step and the Expected Sufficient Statistics (ESS)

Into the Expectation step, instead of describing a distribution in the expected family by the set of parameters, we can describe it in terms of the expected sufficient statistics (ESS). They are two equivalent approaches but we are going to see, in the next chapter, that the computation of ESS simplifies the maximisation step. This concept will be crucial and it will allow us to understand in detail the logical mechanism of the EM algorithm [15].

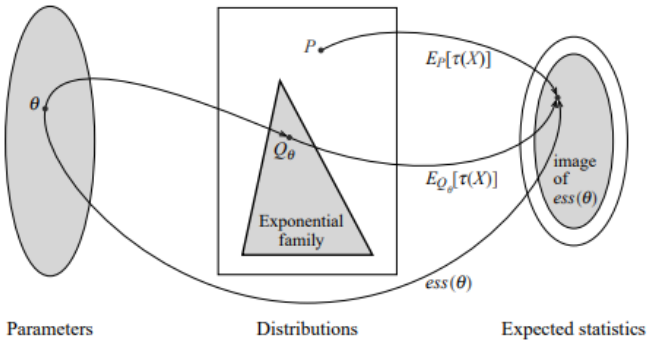


Figure 2.1. Relations among parameters, distributions and expected sufficient statistics

Intuitively, in Figure 2.1 it is possible to show that each parameter corresponds to a distribution, which in turn corresponds to a value of the expected statistics. The function *ess* maps parameters directly to expected statistics. In fact, an expected sufficient statistic is a map from legal parameters θ to vectors of sufficient statistics. In this thesis, we will not go into detail of the theory behind the expected sufficient statistics because many theoretical concepts will need to be introduced as the concept of *M-Projection*. However, it is important to highlight that the algorithm uses the current parameters θ to compute the expected sufficient statistics. In particular, when we learn from complete data, find expected sufficient statistics is a linear task. However, when we are dealing with missing data, we do not have access to complete statistics. In this case, it is possible to see the expectation step how the succession of two steps. Before proceeding with the discussion, it is necessary to clarify the notation. We denote with m an index of a partially observed data case. We indicate with o a partially specified data case while, we denote with $o[m]$ a partially specified data case in the m'th position of the dataset. $H[m]$ denotes the variables whose values are missing in the data instance $o[m]$. The notation $h[m]$ indicates a single variable in $H[m]$. In addition, we indicate with $\xi[m]$ the m'th training example. Then, the E-step includes these two steps:

- The computation of all marginal probabilities:

$$Q(X, U) = P(X, U|o, \theta) \tag{2.5}$$

- The computation of the expected sufficient statistics for all x, u as:

$$\bar{M}_\theta[u] = \sum_{m=1}^{M} \sum_{h[m] \in Val(H[m])} Q(h[m]) I\{\xi[m] < Y >= y\} \tag{2.6}$$

This last formula may be difficult to understand, for this reason, an example at the end of this section will be shown with the purpose to clarify and make more intuitive the EM algorithm. For the moment it is useful to focus our attention on the two summations. To compute the expected sufficient statistics, we must sum over all the completed data cases.

For example, if we have two hidden variables var_1 and var_2, and both the two variables can take 3 different values $\{1,\ 2,\ 3\}$, we must consider 6 different data cases: $< var_1^1, var_2^1 >$, $< var_1^1, var_2^2 >, < var_1^1, var_2^3 >, < var_1^2, var_2^1 >, < var_1^2, var_2^2 >, < var_1^2, var_2^3 >$. In particular, for each $o[m]$, the number of completions is exponential in the number of missing values. Thus, if we have more than few missing values in a instances, an implementation of this approach will not be able to finish computing the expected sufficient statistics. Soon, we are going to see that this concept just described is the definition of EM Soft Assignment and we are going to discuss that there is a better approach to computing the expected sufficient statistic than simply summing over all possible completions. Thus, we are going to define the EM Hard Assignment.
Before presenting EM Soft and EM Hard, it is very important to make a couple of considerations. The logic behind the EM Soft and EM Hard algorithm is the same. There are no difference on the schema presented in the listing above. The only difference concerns how missing values are completed, in EM Hard a single assignment is selected and consequently also the function to compute the Expected Sufficient Statistics changes removing the other marginal probabilities. In addition, it is necessary to remark that the keywords *Soft* and *Hard* are not assigned based on the computational time of the two algorithms (on the contrary we are going to remark that EM Soft is significantly slower than EM Hard). This keywords have been assigned depending on how the missing values are completed.

EM Soft and EM Hard Assignment

As notified a moment ago, EM Soft differs from EM Hard based on the computation of ESS. In particular, as EM Hard algorithm, both methods perform two steps: complete the data using the parameters θ^t and use this data to compute the new parameters θ^{t+1}, where $t+1$ indicates the next iteration.
However, unlike EM Soft, EM Hard selects, for each instance $o[m]$ the single $h[m]$ assignment that maximizes $P(h|o[m], \theta^t)$.

EM Hard can be seen as the optimisation of a different objective function that involves both learning parameters and the task of learning the correct assignment of missing variables. The objective is to maximise the likelihood of the complete data given the parameters

$$\max_{\theta,H} l(\theta : H, D)$$

On the contrary, EM Soft tries to maximise the objective $l(\theta : D)$, considering all possible assignments of missing data.

Looking at the definitions, it is possible to highlight that EM Soft can be seen as the complete version of the EM algorithm and it is the preferable approach because it he takes into account all possible completion. Anyway, EM Soft could be computationally very expensive because, as mentioned before, the possible cases of completion grow exponentially. In the literature is present a more relaxed version of the EM algorithm, called EM Hard. We report the most important differences between the two versions in Table 2.2

	EM Soft	**EM Hard**
Initialization	Random, uniform or any other way	Random, uniform or any other way
Expectation Step	All possible combinations of data are considered	Select a single assignment to missing variables which maximizes the joint distribution
ESS	$\bar{M}_\theta[u] = \sum_{m=1}^{M} \sum_{h[m] \in Val(H[m])} Q(h[m]) I\{\xi[m] < Y >= y\}$	$\bar{M}_\theta[u] = \sum_{m=1}^{M} I\{\xi[m] < X >= x^1\}$
Maximisation Step	Parameters updated based on the expectation step	Parameters updated based on the expectation step

Table 2.2. Difference between EM Soft and EM Hard assignment

To better clarify these fundamental notions, Algorithms 1 and 2 show the pseudo-code of the EM Soft version and the EM Hard version. The goal is to clearly highlight the most important differences.

Algorithm 1: The Soft version of the EM algorithm.

Choose an initial value $\hat{\theta}_0$ for θ.;
while $\left|\hat{\theta}_{j-1} - \hat{\theta}_j\right| < \varepsilon$, *increasing j:* **do**
  $\hat{\theta}_j = \hat{\theta}_{j-1}$.;
  Expectation step: all possible cases of completion of the missing data are computed: $Q(X, U) = P(X, U|o, \theta)$.;
  ESS: We use the marginal probabilities to compute:
  $\bar{M}_\theta[u] = \sum_{m=1}^{M} \sum_{h[m] \in Val(H[m])} Q(h[m]) I\{\xi[m] < Y >= y\}$.;
  Maximisation step: compute the new estimate $\hat{\theta}_j$ given the ESS;
end
Estimate θ with the last $\hat{\theta}_j$.

On the contrary, the pseudo-code of EM Hard presents some relevant differences. In this case the computation of the expected sufficient statistics is relaxed:

Algorithm 2: The Hard version of the EM algorithm.

Choose an initial value $\hat{\theta}_0$ for θ.
while $\left|\hat{\theta}_{j-1} - \hat{\theta}_j\right| < \varepsilon$, *increasing j:* **do**
  $\hat{\theta}_j = \hat{\theta}_{j-1}$.
  Expectation step: all possible cases of completion are computed but it selects the single assignment which maximizes: $P(h|o[m], \theta^t)$.
  ESS: We use the selected marginal probability to compute:
  $\bar{M}_\theta[u] = \sum_{m=1}^{M} I\{\xi[m] < X >= x^1\}$.
  Maximisation step: compute the new estimate $\hat{\theta}_j$ given the ESS;
end
Estimate θ with the last $\hat{\theta}_j$.

Anyway, it is important to remark that EM soft proves to be the most correct approach but, conversely, it can become computationally intensive given that the possible assignments grow exponentially.

EM Forced version

EM Soft is computationally intensive and its convergence is slower than EM Hard. For this reason, the experiments that will be shown in the following chapters, consider also a variant of the EM Soft algorithm. This version is called ***EM Soft Forced*** or simply *EM Forced*. The idea is to limit the number of iterations so as to stop the computation in the same iteration of EM Hard. Thus, EM Forced encapsulates the same logic of EM Soft, also for the E-Step.

2.5 Some properties of the EM algorithm

We have already mentioned some fundamental characteristics of the EM algorithm. Now, we report the most relevant properties in this section in order to facilitate the comprehension of this imputation method.
The most important property of the EM Algorithm is the **convergence property**.

EM algorithm always ensures convergence but:

- It may converge to a local maximum;
- Convergence could be very slow;
- For the Bayesian networks, EM always ensures convergence if and only if all steps are carried out using exact inference.

This property is very relevant for our study [27] because it is the basis for many considerations. In fact, for some time, the implementation of the EM algorithm is carried out through the use of the the approximate inference. As mentioned before, these methods are faster than the exact inference methods. However, we quickly realised that EM exhibits the worst results compared to other imputation techniques. In addition, it is very important to remark that, like the gradient descent algorithm, also in EM, the initialization of the parameters is crucial because, the starting point could influence the convergence to the global maximum. For these reason, in the absence of information, it might be a good idea to try more initializations in order to compare the results efficiently. Anyway, due to the high number of experiments that we are going to perform, we are going to set a uniform distribution for all the variables.
In addition, a relevant characteristic of the EM algorithm is the slow convergence. For this reason, when the convergence is a serious problem, EM Hard is the preferable method. EM Hard makes bigger steps than EM Soft because it assigns the maximum probability to a single case of completion (unlike EM Soft).

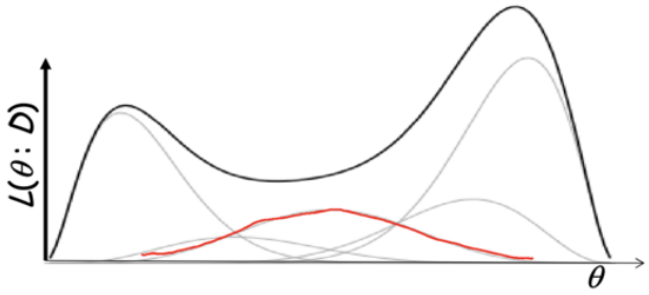


Figure 2.2. Optimization of the objective function

Looking at the Figure 2.2, it is possible to remark that when when we are maximizing an objective function like the one presented previously, we might observe different maximum points on the real parameter distribution. The initial configuration of the parameters may not converge to a global maximum or it may place us on a point where convergence to a local maximum is very slow. For this reason, it is advisable, when we do not know the real distribution of the parameters, to avoid to set the initial configuration of parameters near the extremes values. To alleviate the problem, into the literature, there are different approaches that try to approximate the real distribution of the data as shown by the red line. In this way, a bias is introduced and this approximation can be very far from the actual distribution. Looking at the Figure 2.2 [14], the risk is that the maximum of an approximate function is in fact, the minimum of the real distribution. This could introduce a relevant bias.
Looking at EM Hard and EM Soft it is possible to enunciate the **property of similarity**:

> EM Hard and EM Soft tend to be similar if $P(H|D,\theta)$ assigns a higher probability to one of the possible combinations of data missing during the E-step [15].

In this way, EM Hard and EM Soft tend to the same result. If the variables are balanced then the probability distribution will be uniform. In this case, EM hard and EM soft could lead to different results because there is maximum uncertainty about which values should be assign. This property is well-know in the literature because the presence of an uniform distribution could represent a serious problem. However, this property is only a general result because it ignores some relevant properties of the dataset, the distribution of missing values and the variables and the type of missigness.

2.6 Example

To better clarify the theoretical notions just presented about the EM algorithm, we report an example distinguishing EM Hard version from EM Soft version. The proposed example is simple and it is called *intrusion-earthquake*. Consider the following situation:

You have installed a fairly reliable burglar alarm system in your home, but it also occasionally responds to small earthquakes. There are two neighbours, John and Mary, who promised to phone the workplace after hearing the alarm of the burglar alarm. You also know:

- *John always calls when he hears the alarm ring, but in some cases he confuses the phone ringing with the alarm sound.*
- *Mary likes to listen to loud music and occasionally does not hear the alarm ring.*

It is possible to represents this situation with the following Bayesian network:

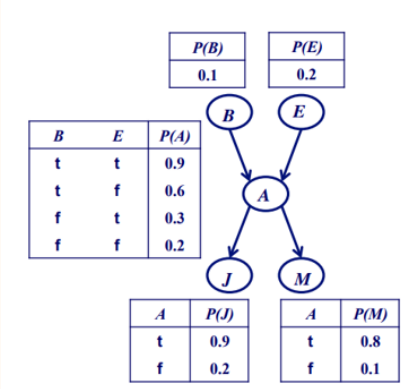


Figure 2.3. Bayesian network associated to the intrusion-earthquake example

In addition, we have the following dataset containing missing values:

	D1	D2	D3
B	?	F	F
E	F	?	T
A	?	?	T
J	F	T	?
M	F	F	F

Table 2.3. Dataset associated to the intrusion-earthquake example

Our goal is to apply the EM algorithm both in EM Hard and EM Soft version to estimate the values of missing values generated through a MCAR mechanism.

Looking at the dataset it is possible to note that we have two missing values in D1 and D2 and an only missing value in D3. Knowing that all attributes are binaries, there are four possible cases of data completion for D1 and D2 and two cases for D3. Let us proceed with

the example by calculating the marginal probabilities.
$Q^1(< B^T, A^T >) = \alpha(0{,}1 \cdot 0{,}8 \cdot 0{,}6 \cdot 0{,}1 \cdot 0{,}2) = \alpha(9{,}6 \cdot 10^{-4}) = 2{,}17 \cdot 10^{-3}$
$Q^1(< B^T, A^F >) = \alpha(0{,}1 \cdot 0{,}8 \cdot 0{,}4 \cdot 0{,}8 \cdot 0{,}9) = \alpha(0{,}02) = 0{,}052$
$Q^1(< B^F, A^T >) = \alpha(0{,}9 \cdot 0{,}8 \cdot 0{,}2 \cdot 0{,}1 \cdot 0{,}2) = \alpha(2{,}88 \cdot 10^{-3}) = 6{,}5 \cdot 10^{-3}$
$Q^1(< B^F, A^F >) = \alpha(0{,}9 \cdot 0{,}8 \cdot 0{,}8 \cdot 0{,}8 \cdot 0{,}9) = \alpha(0{,}41) = 0{,}94$
Where α is the constant of normalization and its value is equal to 0,44.

$Q^2(< E^T, A^T >) = \alpha(0{,}9 \cdot 0{,}2 \cdot 0{,}3 \cdot 0{,}9 \cdot 0{,}2) = \alpha(9{,}72 \cdot 10^{-3}) = 0{,}006$
$Q^2(< E^T, A^F >) = \alpha(0{,}9 \cdot 0{,}2 \cdot 0{,}7 \cdot 0{,}2 \cdot 0{,}9) = \alpha(0{,}023) = 0{,}14$
$Q^2(< E^F, A^T >) = \alpha(0{,}9 \cdot 0{,}8 \cdot 0{,}2 \cdot 0{,}9 \cdot 0{,}2) = \alpha(0{,}026) = 0{,}16$
$Q^2(< E^F, A^F >) = \alpha(0{,}9 \cdot 0{,}8 \cdot 0{,}8 \cdot 0{,}2 \cdot 0{,}9) = \alpha(0{,}104) = 0{,}64$
Where α is the constant of normalization and its value is equal to 0,162.

$Q^3(< J^T >) = \alpha(0{,}9 \cdot 0{,}2 \cdot 0{,}3 \cdot 0{,}9 \cdot 0{,}2) = \alpha(9{,}72 \cdot 10^{-3}) = 0{,}9$
$Q^3(< J^F >) = \alpha(0{,}9 \cdot 0{,}2 \cdot 0{,}3 \cdot 0{,}1 \cdot 0{,}2) = \alpha(1{,}08 \cdot 10^{-3}) = 0{,}1$
Where α is the constant of normalization and its value is equal to 0,011.

After computing the marginal probabilities, we can start the treatment of the EM algorithm, starting with the Soft version and, in particular, with the E-Step. We have two ways to compute the E-Step:

- Derive the Expected Sufficient Statistics using the marginal probabilities as reported in the theory section;
- Using directly the parameters θ and compute an exact inference to estimate the probability distribution of the query variable given the observations.

We are going to show that the two approaches are **equivalent** and lead to the same result. In the literature, however, the first solution is preferable because it is more efficient and it does not require the use of the exact inference through the Enumeration Algorithm. If we use the exact inference to solve the E-Step, our problem aims to calculate:

$$P(X|e) = \alpha P(X, e) = \alpha \sum_y P(x, e, y)$$

For D1
$P(A) = P(A|E = F, J = F, M = F) = \alpha \sum_B P(A, E, J, M, B) =< 0{,}0087; 0{,}991 >$
$P(B) = P(B|E = F, J = F, M = F) = \alpha \sum_A P(B, E, J, M, A) =< 0{,}054; 0{,}946 >$

For D2
$P(A) = P(A|B = F, J = T, M = F) = \alpha \sum_E P(A, M, J, B, E) =< 0{,}22; 0{,}78 >$
$P(E) = P(E|B = F, J = T, M = F) = \alpha \sum_A P(B, E, J, M, A) =< 0{,}2; 0{,}8 >$

For D3
$P(J) = P(J|A = T, B = F, E = T, M = F) =< 0{,}9; 0{,}1 >$

Using the Expected Sufficient statistics, the only component that we have to use are the

marginal probabilities. In this way, we remember that:

$$\bar{M}_\theta[u] = \sum_{m=1}^{M} \sum_{h[m] \in Val(H[m])} Q(h[m]) I\{\xi[m] < Y >= y\}$$

where I is an indicator function, this must be introduced and defined the first time it is used. Suppose we want to calculate $P(E) = P(E|B = F, J = T, M = F)$. We can calculate $\bar{M}_\theta[E^T]$ and $\bar{M}_\theta[E^F]$ very quickly because E is not conditioned by any variable. In this way:
$\bar{M}_\theta[E^T] = Q^2(< E^T, A^T >) + Q^2(< E^T, A^F >) = 0{,}2$
$\bar{M}_\theta[E^F] = Q^2(< E^F, A^T >) + Q^2(< E^F, A^F >) = 0{,}8$
It is possible to show that in this case, we have exploited all the possible cases of completion given by the marginal probabilities to determine the ESS. However, we can describe an important property that we have observed:

$$Q(E^T) = P(E^T|o[m], \theta) = P(E^T|o[2], \theta) = Q^2(< E^T, A^T >) + Q^2(< E^T, A^F >) = 0{,}2$$

This concludes the E-Step of the first iteration of the EM Soft version. before moving on to the M-Step, let us now consider the case of EM hard, starting with a very useful consideration. In the Em Soft we saw that four cases complete the data D1 and D2, while two cases complete D3. In EM Hard we only focus our attention on the single case that maximize $P(h|o[m], \theta^t)$. Intuitively, for D1 we focus our attention on $Q^1(< B^F, A^F >)$, $Q^2(< E^F, A^T >)$, $Q^3(< J^T >)$. Since we have considered the single assignment, we can forget about the other marginal probabilities, in this context we only take into consideration the following table:

	D1	**D2**	**D3**
B	**F**	F	F
E	F	**F**	T
A	**F**	**T**	T
J	F	T	**T**
M	F	F	F

Table 2.4. Data considered for the computation of ESS in the EM hard algorithm

In this way, the function for the computation of ESS changes compared to before because it excludes all the marginal probabilities:

$$\bar{M}_\theta[u] = \sum_{m=1}^{M} \{\xi[m] < Y >= y\}$$

Thus, we have relaxed the ESS function removing a summation. The computation time to compute the ESS becomes linear. However, it is crucial to observe, that EM Hard remains a heavy algorithm because, in the E-Step, it is still necessary to compute all marginal probabilities with the aim to select the optimal assignment to missing variables. Now, we focus our attention on D3. Our goal is to compute the ESS for variable J. We compute

$\bar{M}_\theta[J, A]$ where A is the parent that influence the query variable J:
$\bar{M}_\theta[J^F, A^F] = 1 + 0 + 0 = 1$
Where 1 indicates that we have observed that J assumes value false and A assumes value False. We indicate with 0 the other cases (D2 and D3).
$\bar{M}_\theta[J^F, A^T] = 0 + 0 + 0 = 0$
$\bar{M}_\theta[J^T, A^F] = 0 + 0 + 0 = 0$
$\bar{M}_\theta[J^T, A^T] = 0 + 1 + 1 = 2$

Now, we are in a position to estimate the new parameters using the ESS that we have computed. The M-Step is linear and it is simpler to understand. If we have available the ESS, the estimate of the new parameters is trivial. In fact, it is possible to estimate $\theta_{J^F|A^F}$ simply through the fraction:

$$\theta_{J^F|A^F} = \frac{\bar{M}_\theta[J^F, A^F]}{\bar{M}_\theta[A^F]}$$

It is possible to observe that, in this example, the amount of data is limited and results in EM Hard could be more biased. Usually, the EM algorithm is applied to datasets where the number of data is very large.

Chapter 3

Implementation

In the previous chapter we went into details about the EM algorithm. We analysed the context and the different types of algorithm available in the literature. We observed that EM is a very difficult algorithm to understand and we provided an example in order to clarify some notions. In this way, also the implementation of the EM algorithm is very difficult. For this reason, the implementation of the EM algorithm was a task which required very high levels of attention and, a depth description of the methodological approach is very important. In this chapter we are going to introduce the implementation of the EM algorithm in the Soft, Hard and Forced version.
It is very important to emphasize that at the following link [1] it is possible to access the GitHub repository containing:

- The source code of the EM algorithm in all three versions.
- All the stable code associated with the steps that are going to described in the next Section 3.1.
- The most significant unit tests described in Section 3.2.

3.1 Methodological approach

The entire algorithm is developed using the R language and it requires the import of the package *bnlearn* [20]. The use of this library is essential to handle the BN object. It is important to highlight that the code of the EM algorithm including the E-Step and M-Step has been written from scratch without using any external libraries. Also the execution of the exact inference does not require any library. The development of the EM package starts with the Soft version. Once the implementation is complete, the code of EM soft is used in order to implement EM Hard and EM Forced. The methodological approach defined for the implementation of EM soft follows a bottom-up approach and it is structured according to five steps:

[1] GitHub repository: https://github.com/madlabunimib/Expectation-Maximisation

1. EM Soft works correctly on a very basic example.
2. EM Soft works on very simple BN where all the nodes are binary, and all nodes have at most one parent.
3. EM Soft works on more complex BN. In this step, nodes can have multiple parents. However, all nodes are binary.
4. EM Soft works very well on complex BN with discrete values.
5. EM Hard and EM Forced are implemented.

This is a bottom-up approach because the implementation starts from a very basic example presented in [21] and it evolves increasingly in order to handle more complex Bayesian networks. At the end of each step, automatic and unit tests have been defined in order to validate and verify the correctness of the goals imposed by the specific step.

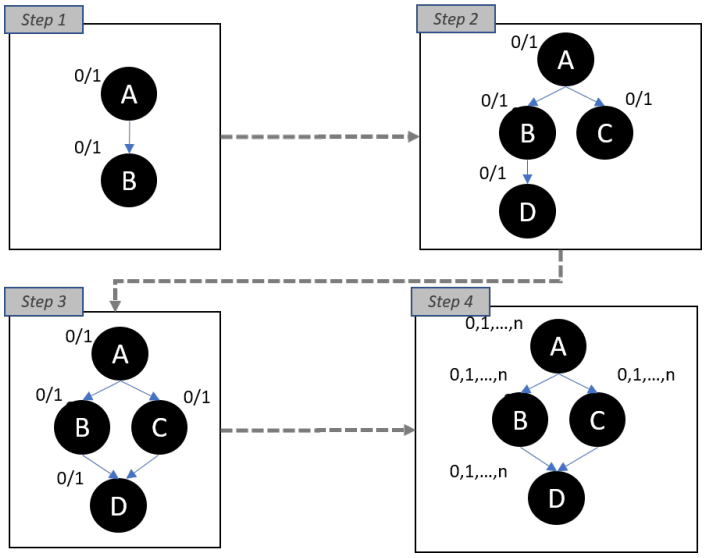


Figure 3.1. Evolution of the complexity of BN as the number of steps increases

Figure 3.1 shows that EM soft is able to process increasingly complex networks during the different phases of the implementation. Writing an algorithm like EM from scratch without a good organization of the work is a extremely complex for different reasons:

- In the literature there are no implementations of this algorithm that use the exact inference and for this reason it is very difficult to compare results. Starting with simple examples allows us to validate results by hand and to detect the unexpected results immediately.
- Given the complexity of the algorithm, it is necessary to debug the code several times with the purpose of check all possible cases and identify any errors. Debug a difficult code makes debugging itself much more complicated.

- It does not allow an easy understanding of the work carried out.
- There are no stable versions. Thorough this approach, the end of each step corresponds to a stable version of the algorithm. If it is necessary, it is always possible to come back to a stable version.

Step 1: EM Soft on simple example

In this fist step, a simple numeric example has been replicated. The BN contains only two binary nodes A e B and an only arcs from A to B exists. The dataset counts ten data and four different missing values. This first step introduces a very important matrix called `M.prob` (marginal probabilities). This matrix is very important because it stores all marginal probabilities useful for the calculation of Expected Sufficient Statistics. It contains also other types of information as: information regarding the complete data useful for the M-Step, the position of missing values useful both for the M-step and E-Step and the observation of the partial data useful for the E-step. In particular, the logic is very simple:

```
1 for (i in 1:10) {
2     M.prob = expectation_step()
3     Data = update_data()
4     cpt = maximisation_step()
5 }
```

Listing 3.1. Iterations defined for step 1

It is possible to note that the code is very simple. In this step, we have not defined the stopping criteria yet. The `expectation_step()` method has two purposes: it computes the marginal probabilities which are stored in the matrix `M.prob` and it exploits the marginal probabilities to compute the ESS. However, it is possible to note that in this implementation, the expectation step includes also the computation of the ESS. This is not an error but only an assumption that we have kept throughout the development. The `M.prob` matrix is very simple but it is going to review in the following steps because on large networks it becomes inefficient: it stores a great amount of information and working with too big matrices is never practical.
The `update_data()` method imputes (updates) the missing values based on ESS calculated previously. Updating the missing values for each iteration is very helpful for debugging and to carry out an analysis of the percentage of correct replacement iteration by iteration with the goal of showing some relevant insights as the overfitting phenomena. In fact, initially, the imputation of the missing values was performed only at the end of the computation. However, due the reasons exposed before, the imputation takes place at the end of each iteration. Eventually, it is important to remark that this method has only the goal of assigning the most probably case of completion to the missing values. The updating step is linear when we deal with EM Hard. When we deal with EM Soft, for each data, we have to compute a *max* function whose purpose is to select the most probably case of completion. The `maximisation_step` method has the goal of computing the new parameters based on ESS. In this case, M-step updates the conditional probability tables of the BN. The maximisation step is linear.

Step 1: Stopping criteria and the early stopping

The introduction of the stopping criteria has anticipated the implementation of the step 2 even if it has only been tested from step 2. The introduction of the stopping criteria is very crucial when dealing with the EM algorithm because, we are going to see, that EM suffers the **overfitting phenomena** from networks with avarage size. EM tends to overfit when it learns very well the distribution of the data but it runs into troubles when it is asked to impute the missing values, exactly as a common Machine learning model. At the end of each iteration, the absolute difference between the previous parameters θ^t and the new parameters θ^{t+1} obtained in the current iteration is calculated. Then, we introduce the first hyper-parameter `ALPHA`. This parameter sets a threshold. If the difference in absolute values between the parameters is minor than `ALPHA`, then the EM algorithm will stop its execution and it produces in output the final CPTs and the complete final dataset. On the contrary, the EM algorithm continues with a new iteration. The final EM procedure as follows:

```
for (i in 1:NUMBER_ITERACTION) {
    M.prob = expectation_step()
    Data = update_data()
    cpt = maximisation_step()
    STOP = stopping_criteria(ALPHA)
    if (STOP){
        cat(sprintf(Exit at the step: %s\n,i))
        break
    }
    else{
        cpt_last = cpt
    }
}
return Data
```

Listing 3.2. Introduction of stopping criteria

`NUMBER_ITERACTION` is the second hyper-parameter and it defines the maximum number of iterations that the EM algorithm can execute without taking into account the condition defined by the stopping criteria. By default this hyper-parameter is set to 20 but it plays a crucial role when we are going to define the EM Forced version. In this case we have to define a strategy in order to stop the execution of the algorithm in a smaller number of iterations (similar to number of iterations requires by EM Hard).
So, this consideration concludes the fist step. We have implemented all the necessary tools for the execution of the EM algorithm. Certainly, several changes have to be done in order to make EM functional on any BN structures.

Step 2 and Step 3: EM Soft for all BN structure's types

In the previous step, an implementation of the EM algorithm was provided. However, it is a very basic implementation because the algorithm works on networks containing only two variables. It is necessary to generalize the code. Step 2 and Step 3 are designed with the goal of executing the algorithm on all types of BN structure. In particular, Step 2 defines

a network with a potentially infinite number of variables but each variable can only have one parent as described in Figure 3.1. In the Step 3, we generalize the Step 2, introducing the presence of multiple parents for a single node. With these two steps, the source code becomes more and more complex and new data structures must be defined in order to make the algorithm work on all possible cases . We have the introduction of three data structures:

- `table_posterior_prob`: is a matrix that is able to store all marginal probabilities for each variable. This structure is updated in the E-step and it is used in the Updating step. This structure differs from *M.prob* described before. In fact, `M.Prob` contains additional and unnecessary information (as the observed variables, the missing values etc.) and it could further increase the computational time. The computational search time using `table_posterior_prob` is almost linear because it contains only the useful information concerning the marginal probabilities.

- `matrix_to_compute`: a structure able of storing any combination of variables and indexing these combinations in their CPTs. The utility of this matrix is very simple: if in an observation there is more than one unobserved variables, these matrix stores all possible combinations among these variables and it also stores an index that allows a linear access to the CPTs of the Bayesian network.

- `compute_probabilities`: this matrix is connected directly to `matrix_to_compute` and it has the same size. We keep separated the matrices in order to not create data structures that are too complex and difficult to use. This matrix is mainly used on the M-step where, `compute_probabilities` stores the partial calculations to be saved later in the CPTs.

An exhaustive example of these structures has been reported within the following presentation[2].
However, to conclude this part, it is important to highlight that the introduction of these three data structures have made it possible to overcome very significant problems that raised when we try to implement the EM Soft algorithm as the presence of a high number of unobserved variables into a specific data.

Step 4: EM Soft on discrete dataset

Through the previous two step, we have obtained a good generalization of the EM algorithm. It works on every type of structures and it is a great result. However, the main limitation is that it works only on binary variables. Actually, it is very rare to work only with binary variables, and, for this reason, extend the algorithm with discrete values is very important. We call γ the unobserved variables such that:

$$P(X|e) = \alpha P(X, e) = \alpha \sum_{\gamma} P(X, e, \gamma)$$

[2]Example associated to the Step 3: https://github.com/madlabunimib/Expectation-Maximisation/blob/master/Presentations/MADLAB_marzo.pdf

The summation sums all possible values of the unobserved variable γ . In the code, the values to which the summation runs are introduced by the matrix `get_not_observed_node`. This matrix is crucial because it facilitates the understanding and the implementation of the algorithm. The access to this matrix is linear and it is used only in the expectation step.

3.2 Automatic and Unit test

One of the most relevant issues of the implementation of the EM algorithm concerns the validation and the evaluation of the correctness of the results. This issue arises because no EM packages are publicly available, assuming some has been developed. For this reason, it is necessary to validate the code in two different ways:

- **Automatic test** that have the goal of checking the correctness and the consistency of a variable, a matrix, a correct application of a formula etc. These tests are implemented using constructs such as *asserts* or Boolean functions. This class of test has the goal to identify unexpected values. For example, asserts have been used to check the correct matching between data values and the corresponding CPTs. Asserts have been used also to check if the matrices involved in the computation of the E-Step have been successfully initialized or updated.
- **Unit test** that has the goal of checking inconsistencies among the results. In this case, a particular test case has to must be specified by defining the following components:
  - The **scenario** taken into account. In this case a partial complete dataset, the structure of the Bayesian network and the initial hyper-parameters have been defined.
  - **The goal of the test** Different questions have been defined and each question correspond in a particular functionality tested.
  - **The expected results**.

  In this case an **inconsistent result** is a result that differs from the expected result. An inconsistent result is different from a **bug**. A bug shows up when a malfunction of a program was found. The most dangerous bugs refer to logical errors. For this reason it is very important to formulate a high number of test cases (also limit cases) and check if a result is plausible. Many times, a logical error is hidden in an unexpected result. In general, all possible inconsistent results have been checked by hand through debugging and different bugs have been solved.

If a unit test is particularly large, a simple log-file in the .txt format has produced in order to check rapidly the status of the code when the inconsistent arise. Overall, eight well-documented Unit tests among Step 2 and Step 3 have been defined. These test are reported into the MadLab repository [3]. In addition, others ten unit tests are implemented

[3]Documented tests: https://github.com/madlabunimib/Expectation-Maximisation/blob/master/Presentations/MADLAB_marzo.pdf

to check specific details of the source code. The definition of these tests have allowed to resolve and identify different errors with the final goal of implementing a correct version of the EM algorithm. Other tests are defined for the step 4.

3.3 Final code and Structure of the project

All the source code in available on the repository linked in the introduction of this Chapter. It is possible to distinguish four different codes: the EM Soft source code, EM Hard source code and EM Forced source code. In addition, it has been added a source code that allows the comparison among the results. Now, let us go into the details of the source code. We implement each version of the EM algorithm has associated a R code. Into these file, the code is organized in methods. Each method implements a functionality. For example, we can distinguish the method that check the validity of the input parameters as the dataset and the BN object, from the method that build the `table_posterior_prob` matrix, the the method that implements the E-Step, the method which implements the M-Step etc. Thus, the code results to be completely modular. The source code is divided into logical sub-blocks, mutually independent, in order to provide greater clarity and to simplify the debugging process. Individual sub-blocks (as the E-Step), can be tested individually before they are then attached to an overall application.
The main method of EM Hard (it is equal to EM Soft) is structured as follows:

```
1  em_hard <- function(initial_data, structure, cpt, ALPHA=0.005,
       NUMBER_ITERACTION=20){
2
3          # Check if all input parameters are correct
4          check_input_hard(initial_data,cpt,ALPHA, NUMBER_ITERACTION)
5
6          # Standardize the dataset to a correct form before
               proceeding with the execution of the algorithm
7          Data = standardize_data_hard(initial_data)
8
9          cpt_last = cpt
10
11         # Creation of the BN
12         bn = custom.fit(structure,cpt)
13
14         # Extractions of all incomplete data (we avoid to cycle the
               whole dataset many times)
15         missing_data = list_missing_hard()
16         matrix.missing.prob = prob_missing_hard()
17
18         # Preprocessing
19         table_posterior_prob = Create_matrix_posterior_prob_hard()
20         table_posterior_prob =
               initialize_matrix_posterior_prob_hard()
21
22         # Same logic as the previous listing
23         for (i in 1:NUMBER_ITERACTION) {
```

```
24          M.prob = expectation_step()
25          Data = update_data()
26          cpt = maximisation_step()
27          STOP = stopping_criteria(ALPHA)
28          if (STOP){
29              cat(sprintf(Exit at the step: %s\n,i))
30              break
31          }
32          else{
33              cpt_last = cpt
34          }
35       }
36      return Data
37  }
```

Listing 3.3. Final main source code associated to EM Hard

in the Listing 3.3 it is possible to observe the structure of the main method of the EM Hard version. It is possible to note that some input parameters for some methods have been omitted in order to improve the understanding of the code

3.4 Optimization, parallelization and computational time estimation

Expectation-Maximization is a very complex algorithm and a poorly optimized code could lead to very high computational time. In particular, optimizing code is critical because the networks can be extremely complex and the number of missing values can be very high. The optimization is necessary especially when we perform the experiments because, as we are going to see in the next chapter, a very high number of executions have to carried out. In this part, two different types of optimization have been introduced:

- **Optimizing** the source code according to the best practices and the guidelines of R [22].
- **Parallelization** of the source code especially when we want to execute EM Soft, EM Hard and EM Forced.

The optimization task was carried out during the experimental phase. For many datasets taken into account for experiments, the code may result not optimized.
To implement the parallelization of the source code, we use the library `Future` [4]. The purpose of this package is to provide a lightweight and unified Future API for sequential and parallel processing of R expression via futures. This package implements sequential, multicore, multisession, and cluster futures. In our case, the use of this library was very simple:

[4]Future documentation: https://www.rdocumentation.org/packages/future/versions/1.17.0

```
1  library("future")
2  plan(cluster, workers = 2, gc = True)
3
4  EM_soft %<-% em_soft(initial_data, structure, cpt)
5  results_hard %<-% em_hard(initial_data, structure, cpt)
6
7  results_forced <- EM_soft$forced
8  results_soft <- EM_soft$soft
9
10 results_gt_hard <- compare_em_with_ground_truth(net, results_hard)
11 results_gt_soft <- compare_em_with_ground_truth(net, results_soft)
12 results_gt_forced <- compare_em_with_ground_truth(net,
       results_forced)
13
14 plan(sequential)
15 ...
```

Listing 3.4. Use of Future library in a generic experiment

It is possible to observe that the code is very easy to understand. $\% < -\%$ is the future assignment operator. It creates and runs a process which the scripts EM Soft or EM Hard are executed according to the plan defined in the line 2. The statement `plan(sequential)` defined in the line 14 is very important because it allows to close the parallel sections and return to a sequential computation.
The optimization and the parallelization step have allowed to lower the computation time required by the execution of the EM algorithm.

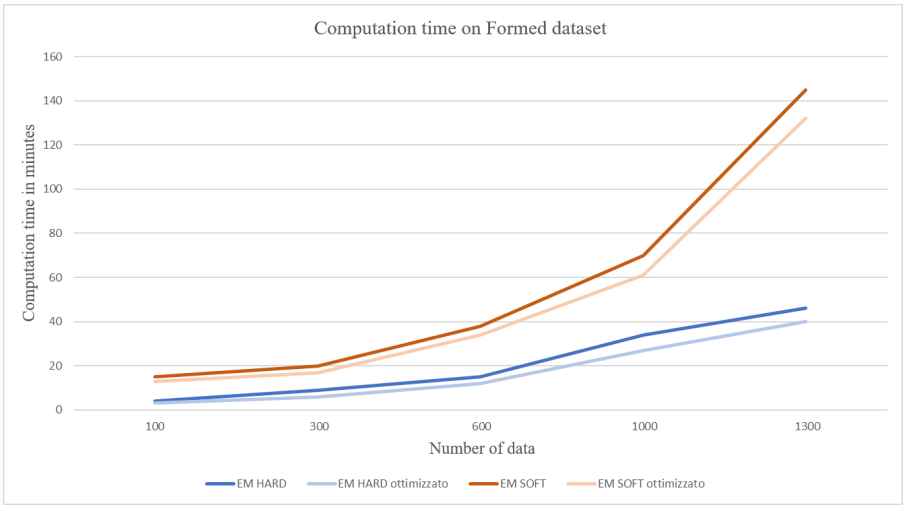


Figure 3.2. Performance of EM before and after the application of the R best practices

These graphs are very important because they begin to give us an estimate of the computational time. It is possible to evidence that the curves tend to an exponential

form when we increase the number of the samples. In particular, looking at Figure 3.2 it is possible to note that the optimization allows to gain a few minutes of computation time. this evaluation was conducted on a large network that is going to be present into details in the next chapter. The dataset taken into consideration consists of 88 different variables and 138 arcs. If we compute EM on 1300 samples, then we will consider 114400 different cells (or values). We set the percentile of missing values to 5%. In this case, the number of missing values on the average will be 5720[±572] (we are going to see in the next chapter that a tolerance of 10% of missing values has been introduced in order to create valid datasets). Looking at this numbers, it is possible to note that it is expensive to apply the exact inference algorithm and the performance tends to be very expensive. Nevertheless, it is still possible to lower computational times. In fact, during the experiments it has been parallelized the execution of the EM Hard algorithm and EM Soft algorithm. Now, we sum the computation time of EM Soft and EM Hard algorithm illustrated in the Figure 3.2. Then, it is possible to present the performance improvements achieved by the parallelizzation process.

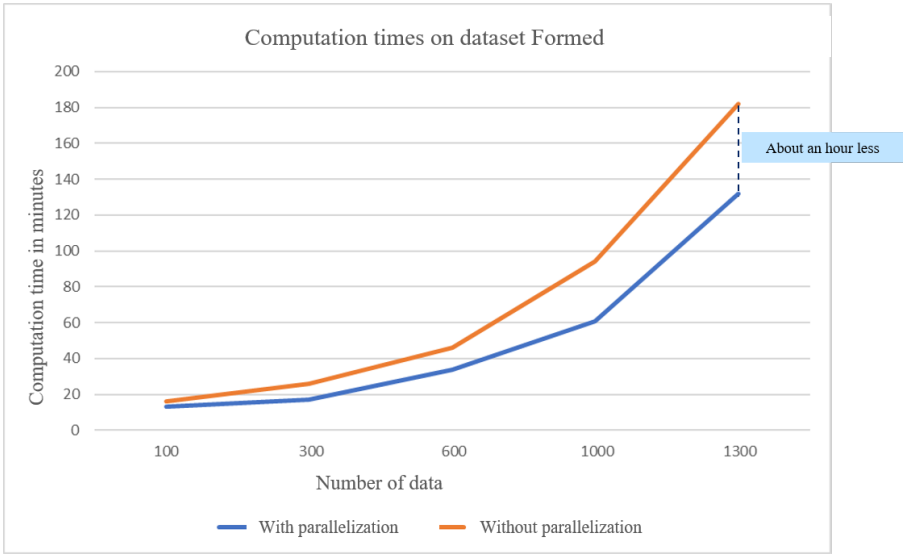


Figure 3.3. EM comparison with and without parallelizzation

In the Figure 3.3 it is possible to highlight that the parallelization has led to a significant contribution. In fact, when we compute 1300 samples (about 5720 missing values), the saving is around one hour. Anyway, in the Chapter 6 we are going to suggest some improvements in order to reduce further the computational time and the memory allocation. Looking at the EM algorithms it is possible to estimate the computation time as follows:

- *Preprocessing step*, which goal is to check the correctness of the input parameters (Bayesian network and dataset), to transform the dataset in a standard form for the execution of the EM algorithm and the creation of the data structures (matrices) useful for the next steps, it takes a polynomial time.

- *E-Step*, which goal is to compute all marginal probabilities to obtain the ESS, it takes an exponential time.
- *Updating step*, which goal is to replace the missing values of a dataset with the most probable value computed in the previous step, it takes a linear time.
- *M-Step*, which goal is to compute the new parameters, it takes a polynomial time.

From this complete discussion emerges that the E-Step is the most intensive step both in terms of computational time and in terms of memory allocation. In future developments Chapter 6, we are going to highlight that optimize more the source code of the EM algorithm is fundamental and we are going to suggest some strategies to do it.

Chapter 4

Experimental results

The experimental phase is fundamental in order to reach our second goal: evaluating and experiment the two versions by varying the complexity of the datasets and adopting different strategies for the generation of missing values. In the previous chapters, we presented and implemented three different versions of the EM algorithm: Soft, Hard and Forced. However, in this moment, it is not clear which is the best version. This doubt is reinforced by the fact that in the literature, a small number of theoretical results are present. So, the experimental part allows us to highlight some insights depending on the properties of the missing data we are considering.
We are going to start the discussion of the experiments by introducing the datasets taken into account. After that, we are going to provide a brief introduction of the formulation of the experiments. At the end, we are going to report and discuss the most important results.

4.1 Evaluated datasets and definitions

Before proceeding with the discussion of the experiments, it is very important to focus carefully the attention on the most properties of the datasets and networks. The study of the properties of each dataset is useful to understand the results with the goal of extracting relevant insights. Now, we provide some useful definitions. After, we are going into the details of each dataset.

Small, medium and large networks

A network is of **small**, **medium** or **large** size if the number of nodes is in the corresponding ranges: [2 nodes; 19 nodes], [20 nodes; 49 nodes], [50 nodes; ∞ nodes] [24].

In this chapter we are going to consider different types of networks. In particular the BN that we are going to be analyzing are described in the Table 4.1:

network	Name	Number of nodes	Number of Arcs
Small networks (2-19 nodes)	ASIA	8	8
	SPORTS	9	15
Medium networks (20-49 nodes)	ALARM	37	46
	PROPERTY	27	31
Large networks (>50 nodes)	HAILFINDER	56	66
	FORMED	88	138
	PATHFINDER	109	195

Table 4.1. Selected datasets for the experiments

All these datasets have been taken by *the The Bayesys data and Bayesian network repository* [7]. Let us continue with the definitions:

Strongly balanced datasets

A dataset is strongly balanced if most variables tend to assume a uniform probability distribution.

These definitions are relevant when we are going to present the datasets. In particular, this last point is directly related to the marginal probability distribution of each node into the BN. Studying this property is relevant because it provides a measure of the uncertainty present in the network, as we saw in the Property 2.5. Looking at the number of missing values, we can provide the following definition:

Frequency of the missing values

A dataset shows a **rare missing values frequency** if the PROP (ratio between missing *cells* and total number of *cells* in the dataset) is less than or equal to 1%.
A dataset shows a **low missing values frequency** if the PROP is greater than %1 but less than or equal to 5%.
A dataset shows a **medium missing values frequency** if the PROP is greater than %5 but less than or equal to 10%.
To conclude, a dataset shows a **high missing values frequency** if the PROP is greater than %10 but less than or equal to 30%.

In our contexts, given a dataset, a *cell* is a the intersection of the i-th row with the j-th column. A cell may contain a discrete value or it may not contain any value. In this last case we say that the cell is **empty**. In real contexts, the datasets that exhibit more than 30% of missing values are not considered because considered not very useful (the uncertainty is too high and it is very difficult to extract useful insights). In this experimental chapter we are going to execute the EM algorithm on all four categories of missing values frequency. Anyway, due to computational time, the treatment of the datasets which exhibit a high missing values frequency will concern only the small-size networks. Now, we provide the last definition:

Missing values distributed uniformly

The missing values are **distributed uniformly** on the dataset if each variable has approximately the same frequency of missing values. On the contrary, a dataset presents missing values on specific variables if the missing values are concentrated on a limited set of variables (e.g. the most connected nodes of the BN, the nodes with the highest outdegree, target attributes, root nodes, leaf nodes, etc...).

This last definition is fundamental because, in order to carry out a complete study about the performance of the EM algorithm, it is important to generate missing values using different strategies, focusing the attention on the nodes which we suppose to be the most important ones.
Let us now turn our attention to the notation. When we generate missing values on all variables, we have used the word *uniform*. Although uniformity is a desired property, this is too strength and it is difficult to obtain when we generate missing values of MAR or MNAR types. In this case we use the adjective *uniform* to describe the fact that all variables contain at least a missing value and the distribution of the missing values tends to be uniform for all the variables. Because using the word *uniform* is misleading, we are going to call these experiments **random patterns**. The term comes from the fact that in order to introduce missing values into the complete dataset D^+, we have to define a square matrix called `pattern` which defines all the rules useful to generate the missing values. In this case the most important rule that must be defined is: *all variables could contain missing values*. In order to reach this property, the main diagonal of the matrix must be set to 0. All the possible values of the matrix will contain 0 or 1 randomly. On the contrary, we are going to relax manually this rule in case we want to generate missing values only on a specific types of variables. In this case we remove the random component because the choice of the partially observed variables has to defined manually according to a specific strategy as, for example, the generation of missing values on the leaf nodes. Anyway, in the Section 4.2 we are going into details of this matrix.
These are the most relevant properties that we are going to consider. In the next section we are going to provide a careful description of the datasets listed in Table 4.1.

Asia

Asia is a small traditional network for diagnosing patients at a clinic. It consists of 8 nodes and 8 arcs. Figure 4.1 shows the true graph of Asia.

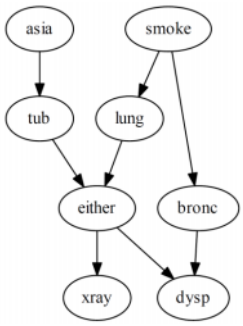


Figure 4.1. True graph of Asia

It is possible to highlight that the maximum indegree and outdegree are both equal to 2. Asia presents two root nodes and two leaf nodes. The structure is very simple. Looking at the dataset, we have focused our attention on the version 5k. This means that the dataset counts 5k different complete samples. Looking at the properties described previously, ASIA is an unbalanced dataset as shows Figure 4.2.

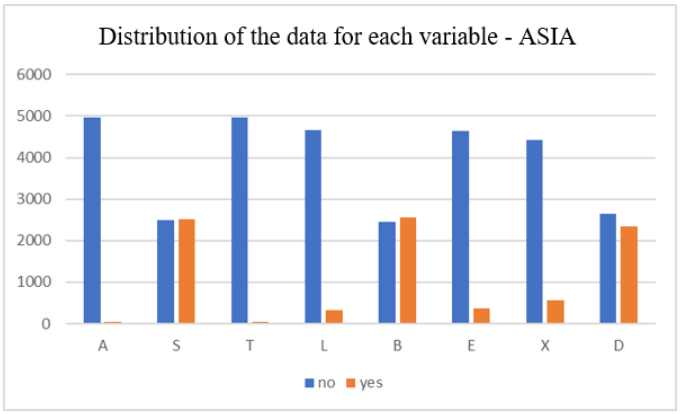


Figure 4.2. ASIA dataset

In particular, only three variables are strongly balanced: *Smoke, Bronc* and *Dysp*; the other variables are strongly unbalanced. Looking at the values, all variables are binaries (each variable assumes only two possible values: true or false).

Sports

Sports is a small BN that combines football team ratings with various team performance statistics to predict various match score outcomes. It consists of 9 nodes and 15 arcs. Figure 4.3 depicts the true graph of Sports.

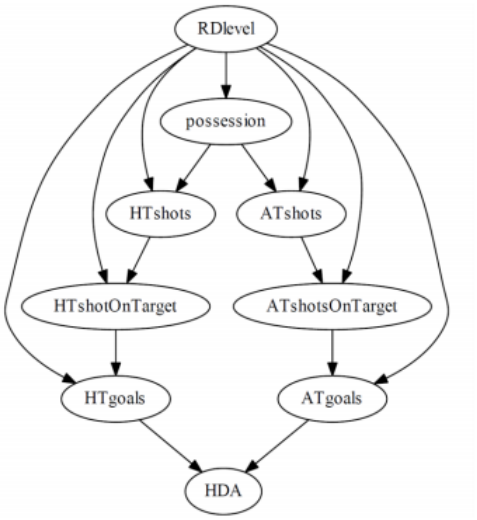


Figure 4.3. True graph of Sports

The network consists of a single root node: *RDlevel*. This node achieves the maximum outdegree: 7. The maximum indegree is equal to 2. There is a single leaf node: *HDA*. This is the target attribute while the goal is to predict the outcome of a football match it can take three possible values: Home (H) team win, away (A) team win or draw (D).
Looking at the dataset, we have considered the version 10k. Sports presents two relevant differences compared to Asia:

- The dataset associated to the Sports network is a strongly balanced dataset as referred in Figure 4.4.

- Unlike Asia, the variables in Sports are not binaries and they take up seven different values.

For this reason, the selection of this dataset is fundamental because it exhibits very significant differences respect to Asia.

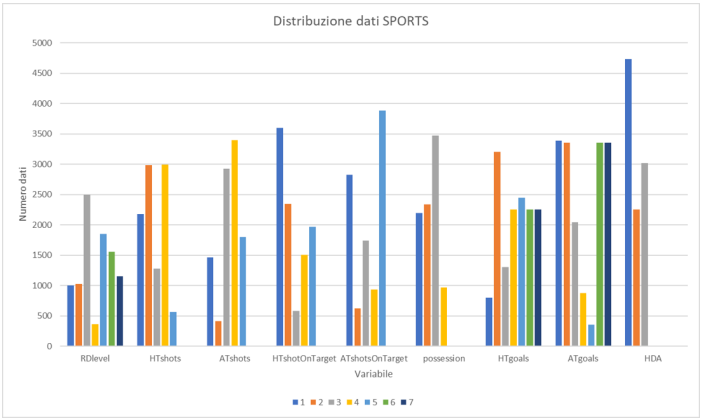


Figure 4.4. Sports dataset

Alarm

Alarm is a medium traditional network based on an alarm message system for patient monitoring. It consists of 37 nodes and 46 arcs. Figure 4.5 depicts the true graph of Alarm.

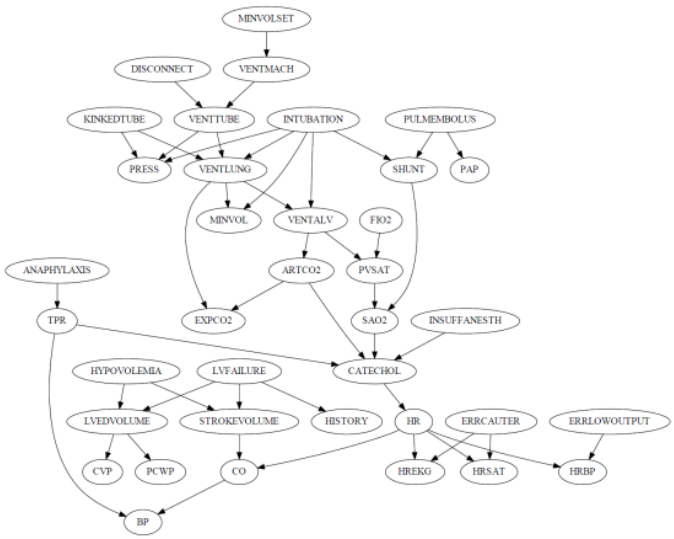


Figure 4.5. True graph of Alarm

It is possible to note that the structure of this network is more complex than the previous two. Alarm has 12 root nodes and 11 leaf nodes. The maximum indegree is equal to 4 while the maximum outdegree is equal to 5. Looking at the dataset, we have considered the version 20k. Such choice results in an unbalanced dataset and in the Figure 4.6 it is possible to highlight that just a small percentage of variables are strongly balanced.

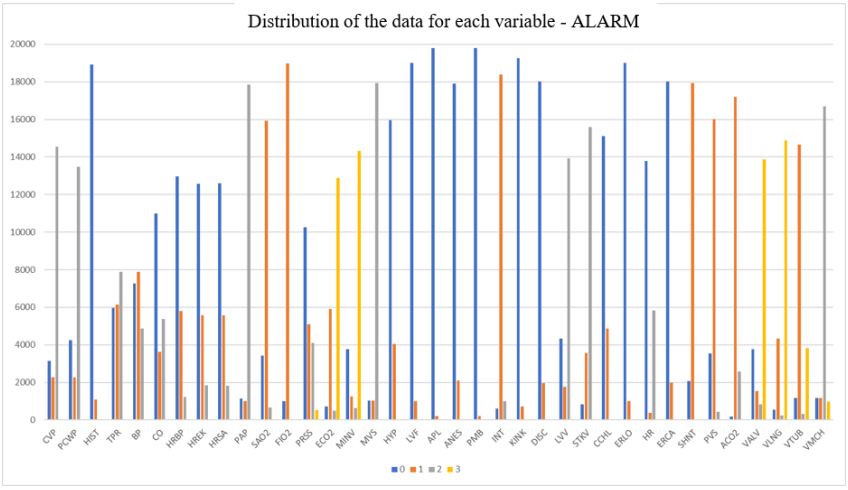


Figure 4.6. Alarm Dataset

Property

Property is a medium BN that assesses investment decisions in the UK property market. It consists of 27 nodes and 31 arcs. Figure 4.7 depicts the true graph of Property.

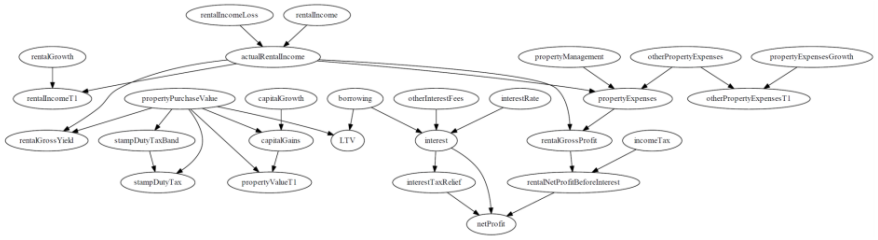


Figure 4.7. True graph of Property

Looking at the Figure 4.7 it is possible to note that the maximum indegree is equal to 3 while the maximum outdegree is equal to 6. The graph consists of 12 root nodes and 7 different leaf nodes. The network presents a single binary attribute. The dataset is strongly balanced and we report the distribution of the values in Figure 4.8. We have selected the version 10k in order to perform the experiments. Comparing this dataset with Alarm it is possible to highlight that this dataset shows very different properties.

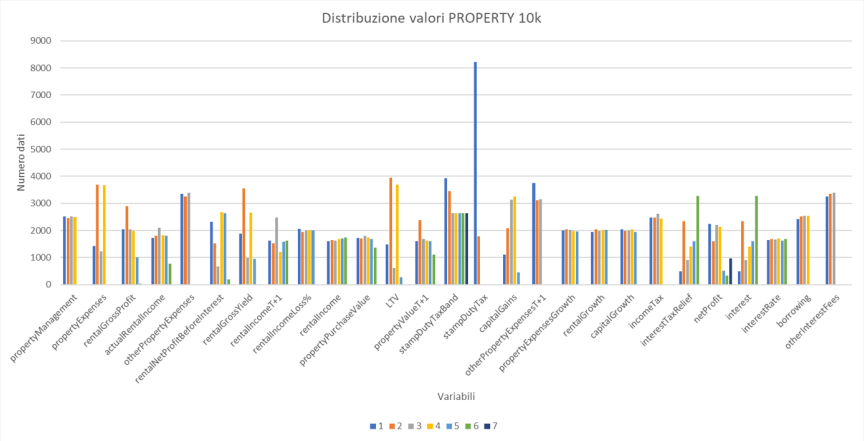


Figure 4.8. Property Dataset

Hailfinder

This is the first of the three large-size networks that we are going to consider in this experimental part. Hailfinder is a Bayesian network designed to forecast severe summer hail in northeastern Colorado. It consists of 56 nodes and 66 arcs. The true graph is depicts in the Figure 4.9.

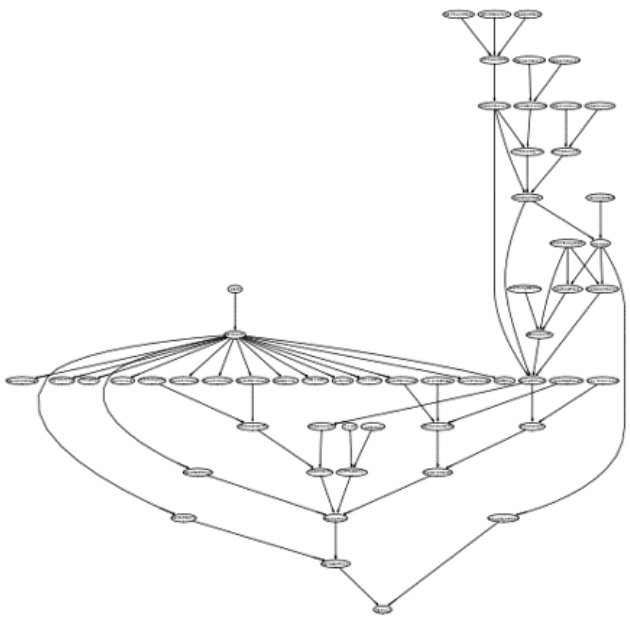


Figure 4.9. True graph of Hailfinder

As far as the true graph is concerned, Hailfinder consists of 13 leaf nodes and 13 root nodes. The avarage degree of Hailfinder is equal to 2,36. The maximum outdegree is equal to 14 and it corresponds with the node called `Scenario`. The maximum indegree of the network is equal to 4. Looking at the dataset, we have selected, for our study, the version 20k. In this case, the dataset consists of 20k data and 1.120.000 cells. Hailfinder results a balanced dataset as shown in the Figure 4.10. Looking at this figure, it is also possible to highlight the complexity of the dataset. Another peculiarity of this network is that the attribute `Scenario`, can assume 11 different values. Hailfinder consists of a single binary attribute.

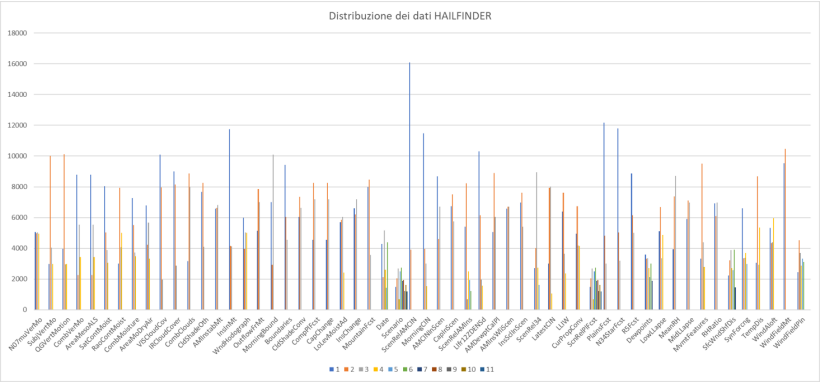


Figure 4.10. Hailfinder Dataset

Formed

Formed is a large BN that captures the risk of violent re-offending of mentally ill prisoners, along with multiple interventions for managing this risk. It is the most complex Bayesian network that we consider in our experiments. In fact, Formed is the network with the largest number of arcs. In particular, Formed consists of 88 nodes and 138 arcs. Figure 4.11 depicts the true graph of Formed.

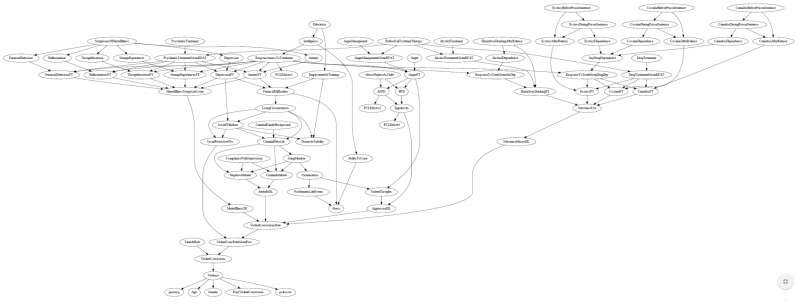


Figure 4.11. True graph of Formed

The maximum indegree and the maximum outdegree of this network are equal to 6. The BN consists of 14 root nodes and leaf nodes. Looking at the dataset, it is possible to highlight a massive presence of binary attributes and many of these attributes are strongly unbalanced. In the experiment, we have selected the version 10k. Figure 4.12 shows the distribution of the values for each attribute.

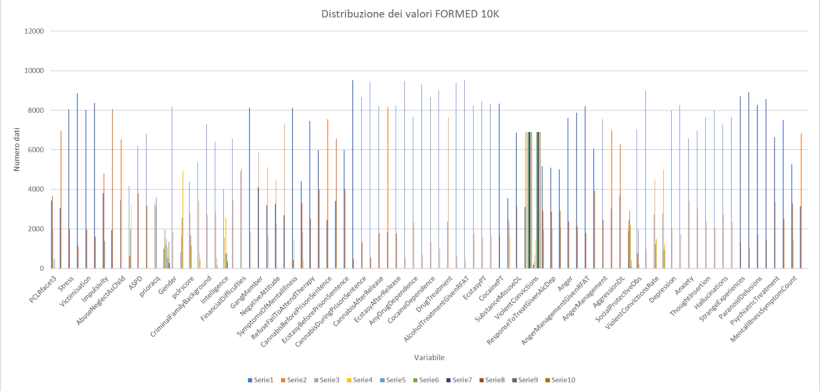


Figure 4.12. Formed Dataset

Pathfinder

Pathfinder is a very large traditional network. It was designed to assist surgical pathologists with the diagnosis of lymph-node diseases. It is the most particular BN that we take into account because, as shown in the Figure 4.13, the structure is very different than the previous networks. In particular, the network shows a only leaf node is called `Fault`. This node exhibits the most higher outdegree.

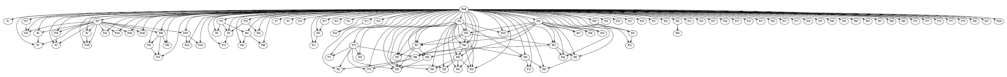


Figure 4.13. True graph of Pathfinder

This network has a high number of peculiarities which require a careful description. In fact, it is possible to note that this network presents a very low avarage degree but the maximum outdegree is the biggest one compared to other networks. The node `fault`, that corresponds to the most central node of the network, consists of 64 different values. Another important peculiarity of this network is that the dataset has missing values natively. This consideration may present different problems. For example, How does the EM algorithm learn the missing values? How does EM distinguishes the naive missing values and the missing value generated by the `ampute` method? To solve this problem, our approach starts with the selection of the version 100k and we have removed all records which had missing values inside. Figure 4.14 shows the distribution of the values for each attribute

of Pathfinder after removing all missing values. In this way, a complete dataset has been obtained. Looking at the Figure 4.14 it is also important to note that the dataset presents a strong unbalance of the values with the exception of the node `Fault` whose values are balanced.

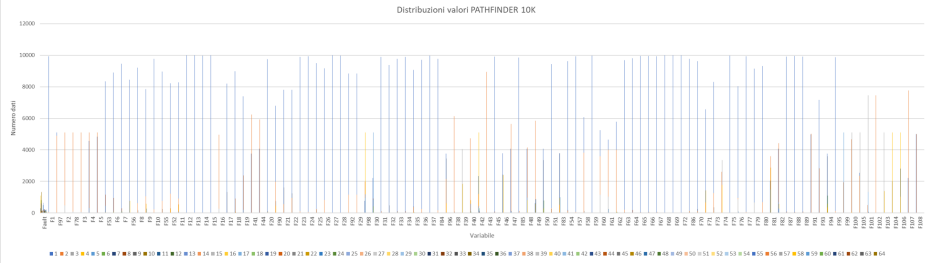


Figure 4.14. Pathfinder Dataset

To conclude this section, we want to emphasize that the dataset and the networks we have chosen are very different among themselves with regard to the properties presented at the beginning of this section. The selection of this datasets allows us to carry out a complete study about the EM algorithm. However, at the moment, we have available only complete datasets, before proceeding with the documentation of the results obtained, it is important to focus our attention on how missing values are generated, how the experiments are performed and evaluated. In the following sessions we are going to enter into details of these issues.

4.2 Generation of missing values

One of the most important issues concerns the definition of the mechanism to generate the missing values. This is not an easy task because the generation of the missing values affects the performance of the EM algorithm. It is important to examine all possible cases in order to conduct a complete analysis:

- According to the **mechanism of missingness**, EM algorithm needs to be tested on all types of missing values: MCAR, MAR and MNAR.

- According to the **amount of missing values**, EM algorithm needs to be tested both on datasets that present a rare/low missing values frequency and on datasets that present a medium/high frequency of missing values.

- According to the **distribution of missing data**, EM algorithm needs to be tested by generating the partial dataset through different strategies such as: missing values generated uniformly among variables, missing values concentrated on the most relevant nodes, nodes with the highest outdegree, target attributes, root and leaf nodes etc.

Therefore, it is important to note that different experiments have to be defined in order to cover all the possible cases. In addition, the properties of the true graphs play a crucial role when we define the experiments. For example, in a small-size network, a study which focuses on examining missing values generated on the most central nodes might not be very significant. However, when we generate the partial dataset, different problems could arise:

- How are the observations selected and the missing values generated?
- How can we define missing values only on specific variables?
- When can a dataset be considered valid?

Selection of the data and generation of missing values

In many of our experiments, we choose the complete dataset in the version 5k or 10k as starting point. However, due to the computation computational requirements and the high number of experiments, we do not execute the EM algorithm on all the 5k or 10k observations. For this reason, it is crucial to define a good strategy in order to select a sample of the dataset.
This is a relevant aspect because, if we choose the observations randomly, we introduce a bias into the results. This happens because, in our subset, the distribution of probability of the variables of the data selected could differ from the actual probability distribution computed considering the complete version of the dataset. Therefore, the sample of observations that we extract from the dataset has to guarantee some properties. In this case, the concept of **replicate** can been introduced. The idea is very simple: we re-execute the experiments n times by varying the sampled observations from the complete dataset. At the end of the process, we have n different results and we compute the avarage and the standard deviation of different performance measures to evaluate and compare different settings. By default, n is set to 8, but, for the small-size networks the values of n is set to 10 in order to obtain more reliable results.
In addition, in order to guarantee that the distribution of probabilities of the sample is not too different from the actual distribution, for each replicate, we generate two different samples. Then, we select the sample which exhibits the smallest Absolute Probability Difference compared to the actual distribution.
Once the strategy is defined, it is important to implement this reasoning. After selecting the sample of observations, `Mice` library is used to generate the missing values from a complete subset and in particular, the `ampute` method [5] can been taken as reference. The `mice` package implements a method to deal with missing data while the `ampute` function generates multivariate missing data in a MCAR, MAR or MNAR manner. When missing values need to be generated, there are two relevant hyper-parameters that have to be defined:

- `Prop`: A scalar value that specifies the proportion of missingness. For example, if we set $prop = 0.1$ in a dataset with 100 cells, on the avarage 10 missing values are generated.
- `bycases`: is a boolean value. If it is true, the proportion of missingness is defined in terms of cases (observations). If it is false, the proportion of missingness is defined in

terms of cells. In our experiments this hyper-parameters is always set to false despite its default value is true.

After the partial dataset has been generated, we can execute the EM algorithm. At the end of the execution, another partial dataset is going to be generated by selecting a new sample of data. However, this is only the first step. We continue our discussion specifying how it is possible to generate missing values only on specific variables.

Distribution of missing values

A crucial task that we have introduced in detail in the previous sections concerns the evaluation of the results when we generate the missing values on a limited set of variables. In this part, we discuss the technical aspects that make the generation of missing values on specific variables possible. In particular, we can differentiate two cases:

- A case in which the generation of missing values involves all variables and the missing values have a uniform distribution for all the variables (we call this case **random patterns**).
- Another case in which the generation of missing values involves only a finite number of variables with similar properties.

In both cases `Mice`, and in particular the function `ampute` provides us a hyper-parameter called `patterns`. This hyper-parameter defines a matrix or data frame of size *#patterns* by *#variables* where 0 indicates that a variable should have missing values and 1 indicates that a variable should remain complete. When missing values need to be set on a finite number of variables, the entire column of all other variables is set with the value 1. On the contrary, if we want to generate the missing values uniformly, we define a matrix (corresponding to the hyper-parameter `patterns`) randomly, where all cells can contain only values 0 or 1. In this case, it is important to guarantee the reflexivity of the matrix and we set each cells of the main diagonal to 0. In this case all variables can contain missing values.
In conclusion, the Listing 4.1 describes the list of the parameters associated to the function `ampute` with the corresponding default values:

```
1 ampute(data, prop = 0.5, patterns = NULL, freq = NULL, mech =
    "MAR", weights = NULL, std = TRUE, cont = TRUE, type = NULL,
    odds = NULL, bycases = TRUE, run = TRUE)
```

Listing 4.1. `Ampute` function

Let us assume the default value for all the input parameters for which we have not provided a detailed description.

Definition of validity

Through the `ampute` method, it is possible to obtain a partial dataset D with the desired properties. Moreover, it is fundamental to verify the quality of this partial dataset in order to check the if the properties have been satisfied. For example, if we have 100 observations and we set the `prop` value to 0.1, we will expect 10 missing values. However, this happens

rarely because it is very difficult that the number of missing values is exactly equal to 10 and, therefore, a tolerance value need to be defined. In order to simplify the discussion, we decided to set this error at 10% regardless of the number of data we are considering. In this case, a dataset is valid if the number of missing values is equal to 10[±1].
Anyway, an other issue could arise. For example, if we want to generate missing values on leaf nodes, we can ask ourselves what is the maximum number of missing values generated on the other nodes. This aspect is very relevant and two different definitions need to be presented.

Missing values uniform on all variables

We set:
Ideal number missing values: $IMV = \#cells \cdot prop$
Error tolerance: $E = IMV \cdot error\_rate$ with $error\_rate = 0.1$
Then, a dataset is valid if the total number of missing values is in the following range $[IMV - E, IMV + E]$.

Example: Suppose the total number of cells in a subset is equal to 9.000, setting prop = 0.01, a dataset is valid if the total number of missing cells falls within the range: [81,99].

Missing values on a limited set variables

We set:
Ideal number missing values: $IMV = \#cells \cdot prop$
Error tolerance: $E = IMV \cdot error\_rate$ with $error\_rate = 0.1$
Then, A dataset is valid if the total number of missing values is in the following range $[IMV - E, IMV + E]$, and, for the considered nodes, the percentage of missing values is greater or equal than 0.9.

Example: Let us assume that the total number of cells in a subset is equal to 10.000, setting `prop` = 0.01, a dataset is valid if the total number of missing values falls within the range: [90, 110]. Moreover, suppose also that we have generated a dataset with 100 missing values and that our goal is to generate the missing values on the leaf nodes, and that A, B and C are the leaf nodes. A dataset is valid if the number of missing values concerning nodes A, B and C is greater or equal to 90.
This example concludes the discussion regarding the generation of missing values. In the next section we are going to get into details of the experiments by discussing the formulation and the methodological approach.

4.3 Formulation of the experiment performed

In this Section we focus our attention on the execution of the EM algorithm through the use of the partial datasets obtained in the previous Section. Given a partial dataset D_1, a **replicate** is an execution of the algorithm EM Hard, EM Soft and EM Forced by using the dataset D_1. Usually, 10 different replicates are computed varying the partial dataset. At the end of the computation the average of the results is computed. We compute also the confidence intervals by setting $\alpha = 0.05$. Due computation time, when we work with

medium and large networks, the number of replicates is set to 8. The logical steps are defined as follows:

- The problem is identified and analysed (e.g. ASIA);
- The problem is formulated. In this part, we proceed as follows:
  1. We set the number of replicates. For example in Asia, we set the number of replicates to 10 because Asia is a small-size network.
  2. We define the concept of **iteration**. A iteration is the execution of n replicates for a specif dimension. For example, in Asia, we can define 6 different iterations for our experiments. In this case, 60 different replicates are introduced. However, we are going to fix this notion later in Figure 4.15.
  3. For each iteration, we set the dimensionality of the subsets. For example we execute the EM algorithm on subsets consisting of: 100,300,500,1000,1500,2000 observations.
- The partial dataset D_i of the i-th replicate is generated as described in the previous sections.
- A replicate finishes when EM Hard, EM Soft and EM Forced finish their computation for the datasets D_1.
- An iteration finishes when for a specific dimension, all replicates are finished.
- The experiments finish when for all dimensions defined, we have computed all replicates and we have obtained the results.

It is possible to summarize these points in the following Listing 4.2.

```
# Set the size of the samples we want to test
num_dati = c(100,200,300,400,500,1000,1500,2000,2500)

# Set the percentile of the missing values to generate
prop = 0.15

# Set the tolerance error useful for the definition of validity
error_prop = prop*0.05

# Set the structure of the network
structure = model2network(...)

# Set the initial cpt of the variables
cpt = list(...)

# we iterate on all samples size
for (size in num_dati){

    # For each size, we compute 10 replicates
    for (rep in 1:10) {
```

```
21
22          # we select the data randomly according to the fixed size
23          initial_data = sample(1:length(data), size)
24
25          # we iterate so that we do not get a valid dataset
26          while (!condition of validity) {
27
28              # We generate the partial dataset
29              partial_dat = ampute(initial_data, other_parameters)
30          }
31
32          # We compute the EM algorithm
33          res_hard = em_hard(partial_dat, structure, cpt)
34          res_soft = em_soft(partial_dat, structure, cpt)
35          res_forced = em_soft(partial_dat, structure, cpt,
               NUMBER_ITERACTION = 4)
36
37          # Compute the KLD divergence
38          KLD_hard = compare_em_with_gt(initial_data,res_hard)
39          KLD_soft = compare_em_with_gt(initial_data,res_soft)
40          KLD_forced = compare_em_with_gt(initial_data,res_forced)
41
42          #save the results in a table
43          results = save_results(KLD_hard, KLD_soft, KLD_forced)
44      }
45
46      # Save results, avarage and stabdard deviation
47      write.csv(results,'nomefile.csv')
48
49  }
```

Listing 4.2. Basic code for the definition of an experiment

In order to better clarify the notion of replicate, Figure 4.15 illustrates the concept of replicate and **iteration**. An iteration is the execution of all replicates for a specific *sample size* (a sample size is the dimensionality of the subsets defined into the experimental strategy). It is important to highlight that each replicate defines a new partial dataset D with different missing values. To conclude, an **experiment** is the execution of all the iterations for the defined strategy. Looking at Figure 4.15, we counts 80 different partial datasets (or replicates) and 8 different iterations (10 replicates per iteration).

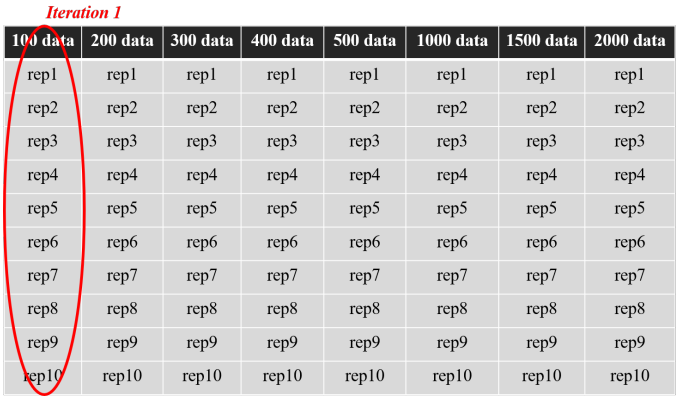


Figure 4.15. Formulation of an experiment

Now, we focus our attention on the performed experiments. Table 4.2 describes for each dataset, the experiments that have been carried out. It is possible to count a huge number of experiments varying the presented properties:

- Size of the network (small, medium, large).
- Distribution of the missing values (uniform on all variables, root nodes, leaf nodes, more central nodes etc.)
- Frequency of missing values (rare, low, medium, rare).
- Number of replicates (10 or 8 according to the network size).
- The number of data (or sample sizes).

To provide an interesting statistic, EM Hard has been run 1480 times in our experiments, 4440 if we count also EM Soft and EM Forced. For this reason, when we are going to discuss the results, we report only a limited number of graphs. In particular, we are going to analyze the most useful and relevant obtained results. Anyway, for the most curious readers we are going to report all the results in the Appendix A.
However, before presenting and discussing this part, in the next section, our attention is going to focus on the evaluation metrics and we discuss how the results were analysed.

Dataset	Network size	Distribution	Frequency of missing values	replicates	Number of data
Asia	Small	Random patterns MNAR e MCAR	low	10	100; 200; 300; 400; 500; 1000; 1500; 2000
			medium	10	100; 200; 300; 400; 500; 1000; 1500; 2000
			high	10	100; 200; 300; 400; 500; 1000; 1500; 2000
Sports	Small	Random patterns MNAR e MCAR	low	10	100, 200, 400, 800, 1200, 1600, 5000
			high	10	100, 200, 400, 800, 1200, 1600
		Most central nodes	low	10	100, 200, 400, 800, 1200, 1600, 2000
			high	10	100, 200, 400, 800, 1200, 1600
Alarm	Medium	Random patterns MNAR e MCAR	low	8	200; 400; 600; 1000; 1500
			medium	8	200; 400; 600; 1000; 1500
		Most central nodes	low	8	200; 400; 600; 1000; 1500
			medium	8	200; 400; 600; 1000; 1500
Property	Medium	Random patterns MNAR e MCAR	low	8	200, 400, 800, 1100
			medium	8	400, 800, 1100
		Most central nodes	low	8	200, 400, 800, 1100
		Leaves	low	8	200, 400, 800, 1100
ForMed	Large	Random patterns MNAR	rare	8	300, 600, 1000, 1400
			low	8	300, 600, 1000, 1400
		Roots	rare	8	300, 600, 1000, 1400
		With major outdegree	rare	8	300, 600, 1000, 1400
		Leaves	low	8	300, 600, 1000, 1400
		Random patterns MCAR	low	8	300, 600, 1000, 1400
		Most central nodes	low	8	300, 600, 1000, 1400
Pathfinder	Large	Random patterns MNAR	rare	8	300, 600, 1000, 1400
			low	8	1000
		Most central nodes	low	8	300,600,1000, 1400
		With major indegree	rare	8	300,600,1000
		With major outdegree	rare	8	300,600,1000
		leaves	rare	8	300,600,1000
		Random patterns MCAR	rare	8	300,600,1000
Hailfinder	Large	Random patterns MNAR	low	8	300, 600, 900, 1200
			rare	8	300, 600, 900, 1200
		Random patterns MCAR	rare	8	300, 600, 900, 1200
		Most central nodes	low	8	300, 600, 900, 1200
		Leaves	low	8	300, 600, 900, 1200

Table 4.2. List of experiments performed

4.4 Evaluation metrics

The discussion of the evaluation metrics is very important in order to interpret the results obtained by the execution of the EM algorithm. We differentiate two different types of analysis:

- Computation of the general results.
- Node-by-node analysis.

In Section 4.1 we have already mentioned these two types of analysis. Anyway, in this Chapter we will go into detail. The first analysis has the goal of extracting a single result, simple to interpret and to compare with the different versions of the EM algorithm. On the contrary, the second approach has the goal of going deeper into the results in order to extract important insights and to justify the cause of the general results.
The experiments are evaluated in terms of:

- Percentile of correct replacement also called PCR;
- Absolute Probability Difference also called APD;

- Kullback-Leibler Divergence also called KLD.

Now, let us go into detail of the evaluation metrics.

Percentile of correct replacement

It is the simplest evaluation metric. In this case, *Percentile of correct replacements* (we call with PCR to abbreviate) compares the missing values replaced by the EM algorithm with the available Ground Truth. In particular, PCR evaluates:

$$PCR = \frac{Number\ of\ correct\ Replacements}{Number\ of\ missing\ values} \tag{4.1}$$

PCR assumes a value in the range [0, 1]. A value closed to 1 indicates that the EM algorithm replaces correctly all the missing values present into the dataset. On the contrary, 0 describes that no missing value has been correctly replaced. PCR has different advantages and different disadvantages:

Advantages	Disavantages
Very simple to interpret	Not always the real data are available
Can be seen as a measure of the accuracy of the EM algorithm	It does not take into account the **probability distributions of the model**: if the probability is uniform, the error must be considered less serious than a probability distribution that tends towards extreme values.

Table 4.3. Advantages and disadvantages of percentile of correct replacement

The second disadvantage shown in Table 4.3 is extremely important: an incorrect classification of a specific variable whose values tend to assume a uniform probability is less serious than an incorrect classification of an other a variable where a value tends to have a probability close to 1. This point makes PCR impractical because it does not provide us a good error indicator.

Absolute Probability Difference

Differently from the previous case, when we compute the *Absolute Probability Difference* (we call APD to abbreviate), the comparison between the results does not involve the actual and the predicted dataset but the comparison involves directly the CPTs of the BN. In particular :

$$APD = \sum_{i=1}^{M} |P_i - Q_i| \tag{4.2}$$

Where P and Q are two probability distribution. In particular P is associated to the distribution estimated by the execution of the EM algorithm while Q represents the actual distribution (ground truth). APD assumes a value in the interval $[0, N]$ where N is the number of probabilities of the CPTs. A value closed to 0 indicates that there are not

differences between the probability distribution. To facilitate the comprehension of this evaluation metric, a normalized version (APDN) has been introduced. In this case:

$$APDN = \frac{APD}{N}$$

This values is simpler to interpret than the previous one because it assumes a value in the range [0,1].
APD and APDN overcome the disadvantages presented with the discussion of PCR. When we carry out a node-by-node analysis, we can easily take into account one variable at a time and compute the APD or APDN on this specific variable. However, APD and its variant APDN suffer a limitation: they don't allow to evaluate very well the information loss which is obtained through the execution of the EM algorithm.

Kullback Leibler Divergence

A good metric that allows us to evaluate the information loss is the *Kullback Leibler Divergence* (we call KLD to abbreviate) [4]. It is a technique deriving from information theory, In the literature, Kullback Leibler Divergence is often called with the term *Kullback Leibler Distance*. However, it is important to highlight that KLD is not a distance metric because it is asymmetric. This means that a divergence is a scoring of how one distribution differs from another, where calculating the divergence for distributions P(x) and P(y) would give a different score from P(y) and P(x). In particular:

$$KL[P(y)||P(x)] = \sum_{i}^{N} P(y_i) log \frac{p(y_i)}{p(x_i)} \tag{4.3}$$

$$\neq$$

$$KL[P(x)||P(y)] = \sum_{i}^{N} P(x_i) log \frac{p(x_i)}{p(y_i)} \tag{4.4}$$

Now, we fix the notations. We call $\mathbf{p(x)}$ a probability density vector derived from the execution of the EM algorithm and we call $\mathbf{p(y)}$ a probability density vector associated with the actual data distribution. Then the correct interpretation of the KLD metric is the formula reported in Equation 4.3.
In many of our experiments we are going to show the results only through this metric. It is important to note that KLD divergence can be used to measure the divergence between discrete and continuous probability distributions, where in the latter case the integral of the events is calculated instead of the sum of the probabilities of the discrete events. However, in our study we focus the attention only on discrete variables.
This is the conclusion of the methodological approach adopted for our experiments. In the next Section we start to report and discuss the most important results. As mentioned previously, due the huge number of results, we are going to limit the treatment only to the most relevant results.

4.5 Analysis of the results

In this section, we report the most important results obtained by the execution of the three versions of the EM algorithm. The goal is to determine if there are statistically significant differences among the versions. We compute the average and the standard deviation in order to identify differences into the results. In particular, the variance measures how far a set of numbers is spread out from their average value. The smaller is the variance, the more the values of the variable are concentrated around the average value. This is an important definition because we prefer the algorithm that exhibits the smallest variance. We remember that the standard deviation is the square root of variance. In particular, it is possible to provide two different definitions that are very useful for our discussion.

Two results r_1 and r_2 show **statistically significant differences** if, looking at the confidence intervals, r_1 and r_2 exhibit different averages and their confidence intervals are not overlapped.

The result r_1 is **better** than the result r_2 if r_1 and r_2 show statistically significant differences and r_1 exhibits the greater average if we consider the percentile of correct replacements, the lower average if we consider the Kullback-Leibler divergence or the Absolute Probability Difference.

The result r_1 is **preferable** than the result r_2 according to the standard deviation, if they exhibit a similar avarage and the confidence interval of r_1 is smaller than r_2

To better clarify these notions, let us give you an example. Figure 4.16 reports three different cases labelled with: *Case A*, *Case B* and *Case C*. For abbreviation, we simply indicate the letters: A, B and C. It is possible to highlight that A shows statistically significant differences than B and C in terms of percentile of correct replacement and it is better than the other two results. Looking at B and C, they exhibit a similar avarage. Anyway, the confidence intervals described by B is smaller than C. For this reason B is preferable to C. In the general case, when we compare two different distributions, the distribution with the greater variance is the most risky [8].

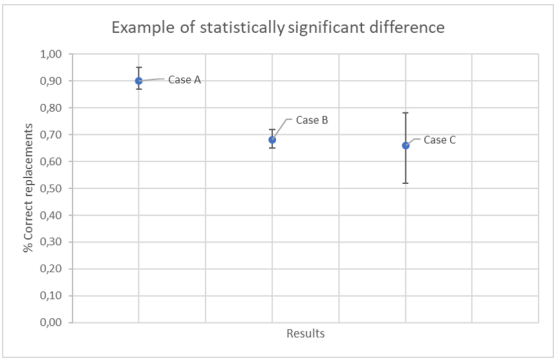
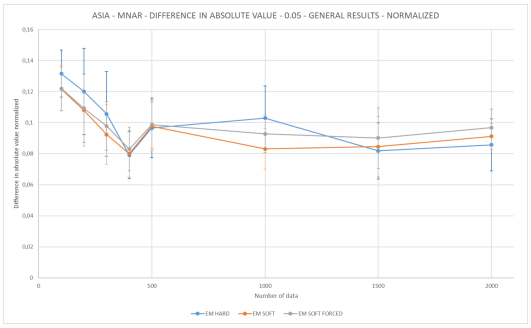
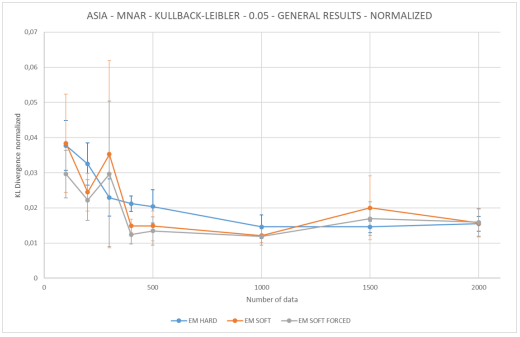


Figure 4.16. Difference between statistically significant differences and differences

4.5.1 Results with Asia

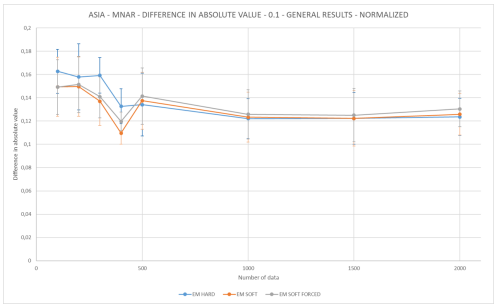


a ASIA - *prop* = 0.05 - Absolute Probability Difference normalized - MNAR

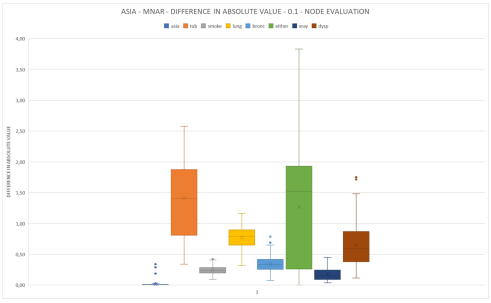


b ASIA - *prop* = 0.05 - Kullback-Leibler normalized - MNAR

Figure 4.17. ASIA - *prop* = 0.05 - MNAR



a ASIA - *prop* = 0.01 - Absolute Probability Difference normalized - MNAR



b ASIA - *prop* = 0.01 - Node evaluation - MNAR

Figure 4.18. ASIA - *prop* = 0.01 - with node evaluation - MNAR

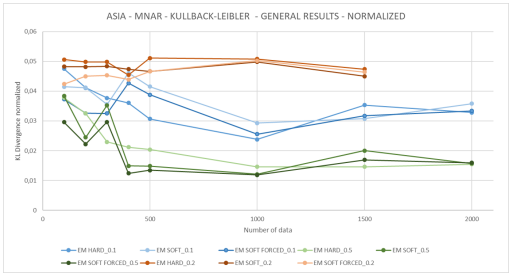
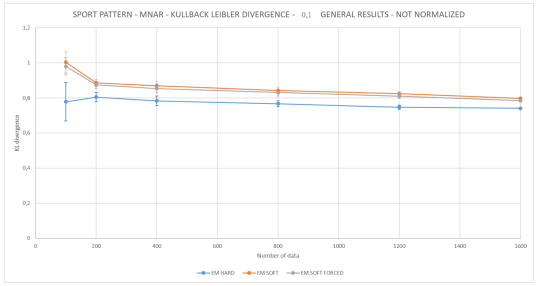


Figure 4.19. EM comparison using KLD divergence varying prop

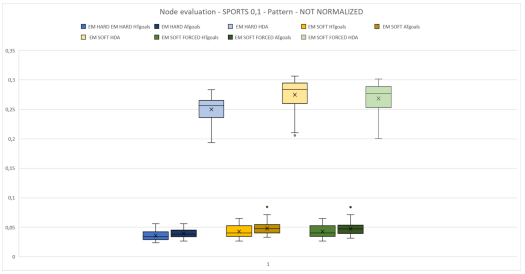
Discussion of the results

Looking at the general results, ASIA does not show statistically significant differences among the versions: EM hard, EM Soft and EM Forced. This is the first general consideration that emerges by observing the line plots. In particular, as presented in Figure 4.2, the dataset associated to ASIA is very unbalanced and, we have observed during the during the execution of the experiment, that EM Hard and EM Soft tend to replace the missing values in the same way from the end of the second iteration.
Looking at the results, it is possible to highlight that when we work with a small amount of observations (in this case less than 500), EM shows a high standard deviation in the results. More precisely, EM exhibits different ups and downs with very large confidence intervals. This phenomena is more visible in the Figure 4.17 and it happens when the number of missing values is too low. In this case EM is constrained to learn on a limited number of data and the results strongly depend on the data selected in the samples. Performance begin to stabilise, also in terms of standard deviation, when the number of missing values begin to grow.
Figure 4.18 (b) shows an interesting result also visible in other experiments. E and T are the nodes that EM learns with more difficulty. This divergence begins to decrease when we increase the number of iterations on the EM algorithm. We remark that due to computational time we have set a maximum of three iterations for EM Hard/EM Forced and a maximum of six for EM Soft.
Figure 4.19 shows a comparison between the three algorithms. In this case, each color represents a different `prop`. It is possible to note that when the percentile of missing values is high, also the divergence tends to be high. This is a expected behaviour because when we set a high `prop`, we have available a smaller number of complete observations. In this case, EM makes more difficult to get closer to the actual probability distribution. In addition, Figure 4.19 confirms that EM Hard, EM Soft and EM Forced do not present statistically significant differences in results.

4.5.2 Results with Sports

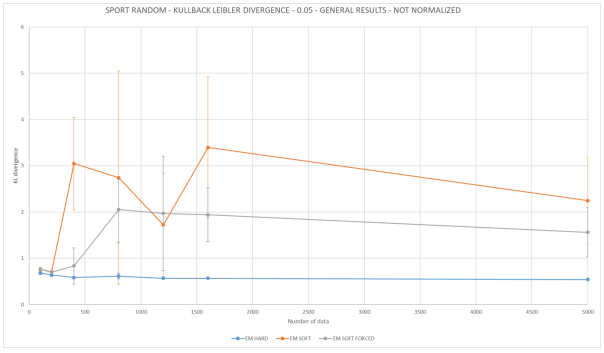


a SPORTS - *prop* = 0.1 - KLD - More central nodes

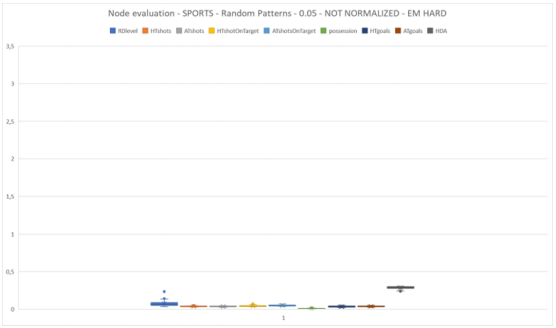


b SPORTS - *prop* = 0.1 - KLD - More central nodes - Nodes evaluation

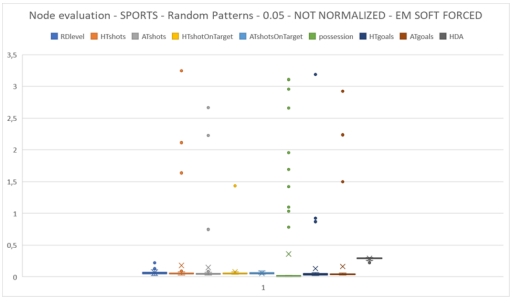
Figure 4.20. SPORTS - *prop* = 0.1 - More central nodes



a SPORTS - *prop* = 0.005 - KLD - MNAR



b SPORTS - *prop* = 0.005 - KLD - EM Hard-Nodes evaluation



c SPORTS - *prop* = 0.005 - KLD - EM Hard-Nodes evaluation

Figure 4.21. SPORTS - *prop* = 0.005 - MNAR (Random Patterns)

Discussion of the results

Looking at the Figure 4.20, EM Hard and EM Soft tend to learn the same distribution of probability and they do not exhibit statistically significant differences although EM Hard is the preferable version. In particular, when we execute the EM algorithm by generating the missing values on the most central nodes (selected based on the degree of each node), all algorithms fit these nodes about the same way (b). The same result is obtained when we reduce the `prop` value.
On the contrary, significant differences are shown when we generate missing values uniformly on the dataset. This is the case of the Figure 4.21. EM Hard tends to have a stable trend with more pronounced standard deviation. Vice versa, EM Soft and EM Forced exhibit the worst results. In this case we prefer the EM Hard version.
When we analyse the results with EM Soft and EM Forced setting the `prop` $= 0.05$, two different patterns arise:

- Up to 1500 observations, EM Soft and EM Forced show an unpredictable trend with different peaks. In this case, the results strictly depend on the data selected in the sample.
- Starting from 1500 data, results begin to stabilise and they exhibit a lower standard deviation. In this case the KLD divergence tends to decrease when increase the number of the data. The best result is obtained when we use all 5000 observations. In this point, the difference in terms of performance with EM hard is the minimum of the experiment. It is very important to note that EM Soft (7 iterations) continues to show worse results than EM Forced (3 iterations). This is a general phenomena because when we increase the number of the iterations, the results tend to worsen.

In particular, looking at the Figure 4.21 (b and c) it is possible to observe a comparison between the results of EM Hard and EM Forced. It is possible to note that EM hard has fitted very well all variables. On the contrary, EM Forced presents a considerable number of outliers (i.e. values that deviate from the average). After analyzing all results, the phenomena illustrated in the Figure 4.21 can be explained through *overfitting*: EM Soft and EM Forced fit very well the probability distribution of particular subsets and in special cases, they cannot replace the missing values creating several outliers. We recall that EM Soft and EM Forced are the versions of the EM algorithm that take into account all possible assignments to missing variables during the computation phase of the Expected Sufficient Statistics. In this case, it is possible to conclude that EM Hard is the best algorithm.

4.5.3 Results with Alarm

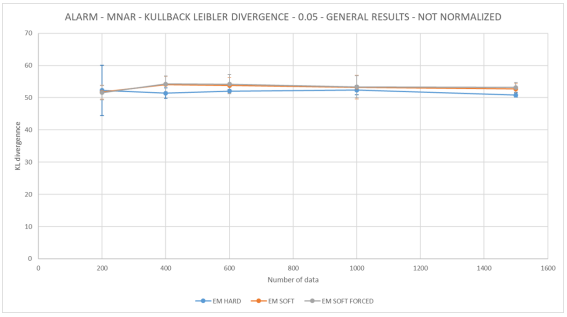
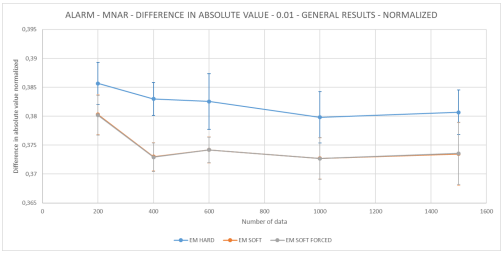
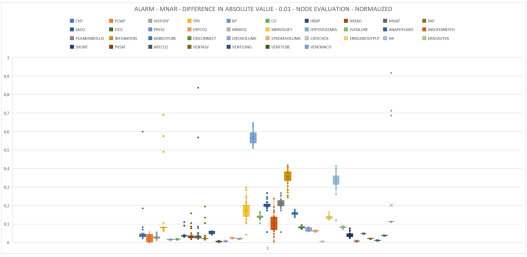


Figure 4.22. ALARM - $prop = 0.05$ - General results



a ALARM - $prop = 0.01$ - KLD normalized - General results



b ALARM - $prop = 0.01$ - KLD normalized - Nodes evaluation - EM Hard

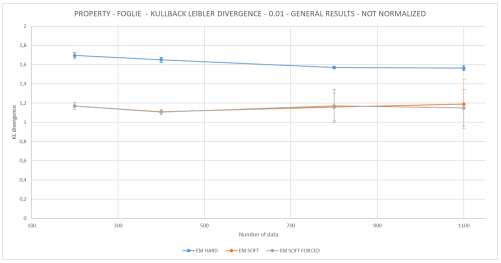
Figure 4.23. ALARM - $prop = 0.01$ - General results

Discussion of the results

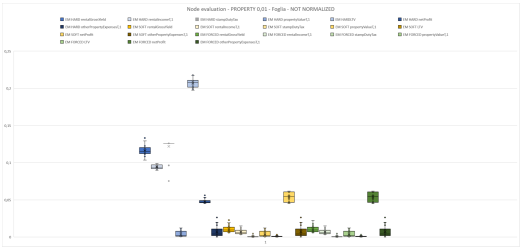
In the general case, there are no statistically significant differences between the EM versions. In Particular, a slight difference is shown in Figure 4.23 (a). In this case the divergence of EM Hard results to be greater than the other two versions and EM Soft and EM Forced result to be the preferable versions. However, this is only a partial result because, increasing the number of iterations of EM Hard from 3 to 6, we are able to achieve the same performance comparing with EM Soft. To reinforce this point, Figure 4.23 (b) shows the nodes evaluation of the Hard version of the Expectation-Maximisation algorithm. It is possible to observe that the most central nodes of the network have not been fitted very well.
On the contrary, Figure 4.22 shows the results reducing the percentage of missing values. In this case, it is shown a typical behaviour of EM when we work with the Alarm network. Looking at this results it is possible to conclude that the versions of the EM algorithm are equivalent.
In general, when we work with Alarm, the same considerations obtained from the analysis

of the results with the Asia network result to be valid. In particular, there is not a preferable version if we increase the number of iterations of EM Hard

4.5.4 Results with Property



a PROPERTY - *prop* = 0.01 - KLD - leaf nodes



b PROPERTY - *prop* = 0.01 - KLD - Nodes evaluation - comparison

Figure 4.24. PROPERTY - *prop* = 0.01 - Leaves

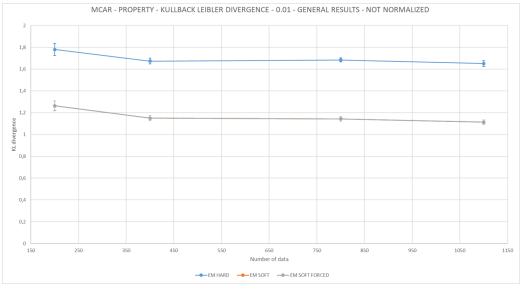
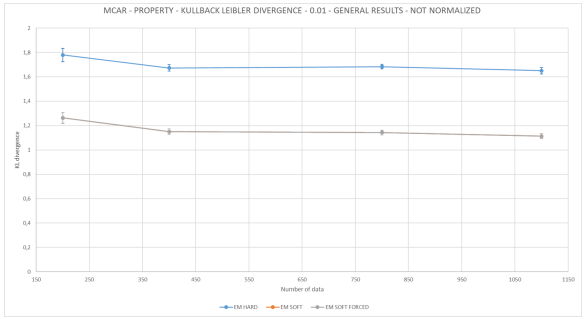


Figure 4.25. PROPERTY - *prop* = 0.01 - More connected nodes

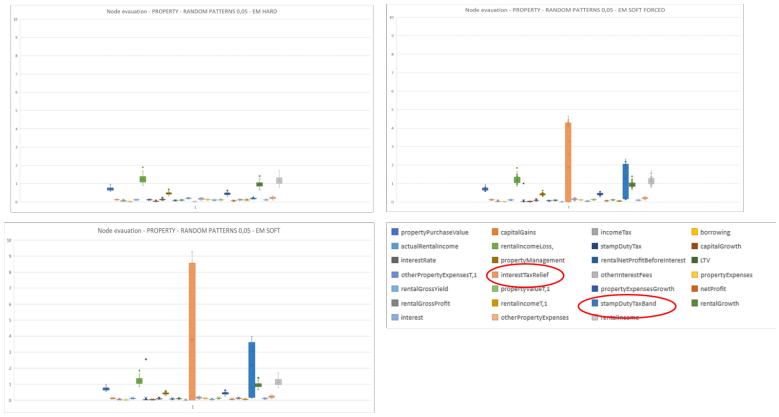


a PROPERTY - $prop = 0.01$ - KLD

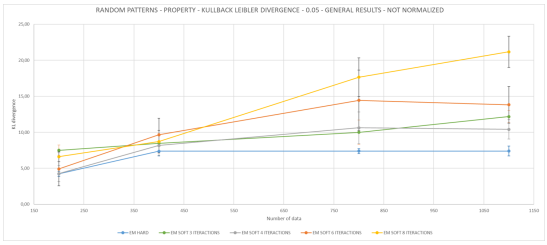
b PROPERTY - $prop = 0.01$ - KLD - Nodes evaluation - comparison

Figure 4.26. PROPERTY - $prop = 0.01$ - MCAR

a PROPERTY - $prop = 0.005$ - KLD



b PROPERTY - $prop = 0.005$ - KLD - Nodes evaluation - comparison



c PROPERTY - $prop = 0.005$ - KLD - proof of overfitting

Figure 4.27. PROPERTY - $prop = 0.005$ - Random patterns

Discussion of the results

Unlike the previous cases, the treatment of the results obtained by the Property network are more difficult to analyze. In particular, we have already described that Property consists of a strongly balanced dataset and the network described in Figure 4.7 is more complex than the previous considered networks.
Let us start the treatment of the results with the leaf nodes. Looking at the Figure 4.24 it is possible to observe that the results in terms of Kullback-Leibler Divergence are the worst when we work with EM Hard. In particular, the divergence describes by EM Hard is significantly greater taking into consideration also the confidence intervals. In general, it is possible to conclude that EM Soft and EM Forced are the best versions and they do not

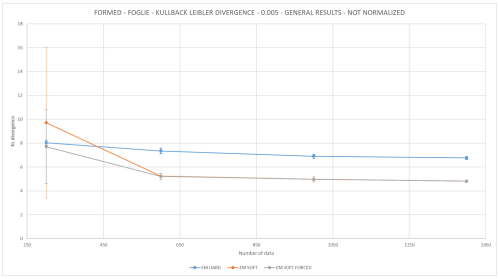
exhibit significant differences. Figure 4.24 (b) focuses the attention on each leaf node. With the shades of blue we report all variables fitted by EM Hard, with the shades of yellow the variables fitted by EM Soft, and with the green color, we report the variables fitted by EM Forced. By looking at the figure, it is possible to highlight a high number of variables which have been fitted in a wrong way by EM hard. The same variables are instead fitted very well from the other two algorithms. Since it is necessary to investigate the reasons of this result, we repeated the experiment increasing the maximum number of iterations of the EM Hard algorithm. Anyway we have observed that, the contribution of this change is minimal (especially when we fit the node called `otherPropertyExpensesT,1`). In particular, it seems that EM Hard has converged with a local maximum that is different compared to the other two EM versions. This result was also achieved when we took into consideration the most central nodes. This experiments is displayed in Figure 4.25. Also in this case, EM Soft and EM Forced are the best versions. A very interesting observation is that, also in this case, the divergent nodes are associated to the leaf nodes as already shown in the previous graphs. *In these contexts EM Hard could not effectively learn the uniform distribution of the data and EM Soft could guarantee more reliable results.*
Now let us focus our attention on the case in which missing values are generated on the entire dataset. Figure 4.26 and 4.27 treat two different cases which lead to two completely different results. The evaluation of these experiments is very important in order to extract very useful insights. The results described by these two experiments are confirmed by other similar experiments. In conclusion, we can report these two cases as reference examples to provide a complete discussion of the results.
When we work with a not rare number of missing values, as reported in Figure 4.26, the EM Hard divergence results greater than EM Soft. In this situation the considerations discussed with the leaf nodes result to be valid and, also in this case, EM Soft and EM Forced are the best algorithms. In fact, In the Subfigure 4.26 (b) it is possible to evidence a leaf node called `otherPropertyExpensesT,1` which exhibits strong differences compared to EM Soft. The same applies to the node called `stampDutyTax` with the blue color. When we increase the number of iterations, the performance of EM Hard improves. However, the EM Hard results are not similar to the results obtained with the execution of EM Soft.
Surprisingly, the discussion changes completely when we evaluate a dataset that presents a rare frequency of missing values. It is the case presented in the Figure 4.27. In this case, differently from the previous one, we have generated the missing values through a MNAR mechanism. We can observed that, with reference to the Property dataset, there are no difference among the type of missigness when we compare the three versions of the EM algorithm. *The relevant aspect that changes is the number of missing values.* In this case it is possible to note a stable performance of EM hard, with very small standard deviations. On the contrary, EM Soft and EM Forced exhibit an increase in KLD divergence when the number of data increases. In addition, EM Forced (which ends its computation before EM Soft) shows statistically significant differences in performance compared to EM Soft. Therefore, it is possible to remark that EM Hard is the best version. Looking at the Subfigure 4.27 (b), it is possible to highlight an odd behaviour of EM Soft and EM Forced in two nodes: `interestTaxRelief` and `stampDutyTaxBand`. Also in this case it is extremely important to investigate the reasons that led to these results. In particular, we can observe that:

- On average, EM Hard has concluded the computation at the third iteration, EM Forced at the fourth iteration, while, EM Soft at the sixth (6.5) iteration.

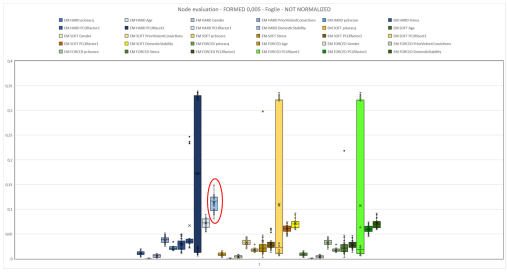
- Looking at the distributions of the variables: `interestTaxRelief` and `stampDutyTaxBand`, it is possible to observe that these variables presented a strong balance of values. In addition, the first variable can take 7 different values, while the second 6.

Therefore, we can formulate the following hypothesis: **overfitting**. In particular, our idea is that by setting the number of iterations very high, EM Soft learns the uniform probability distribution of variables too well, and, in some situations, it is not able to replace missing values correctly. This phenomenon has worsened in EM Soft which computes at least two iterations more than the Forced variant.
The following **experimental setup** must be defined in order to validate the hypothesis: The experiment is re-executed by varying the number of iterations, in the first case by forcing the number of EM Soft iterations to 3, in the second case by forcing the number of iterations to 8.
This procedure is always valid because in our **interpretation**: we expect a high divergence in results when we set the number of iterations to 8. The final results are available in Subfigure 4.27 (c). It is possible to highlight that the yellow line (corresponding with the EM Soft with 8 iterations) tends to grow as the number of observations increases. However, EM Hard remains the favourite algorithm in any case. Looking at the results, it is possible to conclude that our overfitting hypothesis is valid.

4.5.5 Results with Formed

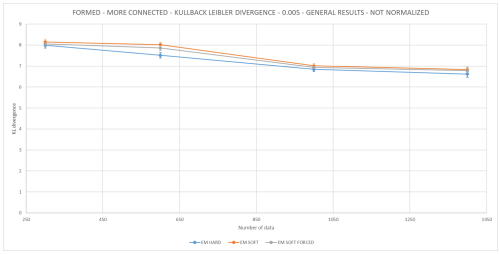


a FORMED - *prop* = 0.005 - KLD- leaf nodes

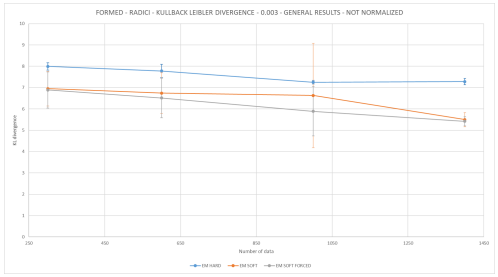


b FORMED - *prop* = 0.005 - KLD- Nodes evaluation - comparison

Figure 4.28. FORMED - *prop* = 0.005 - Leaves

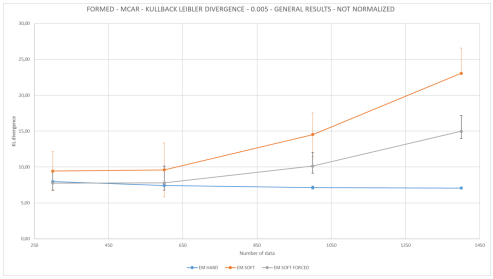


a FORMED - $prop = 0.005$ - KLD- More connected nodes

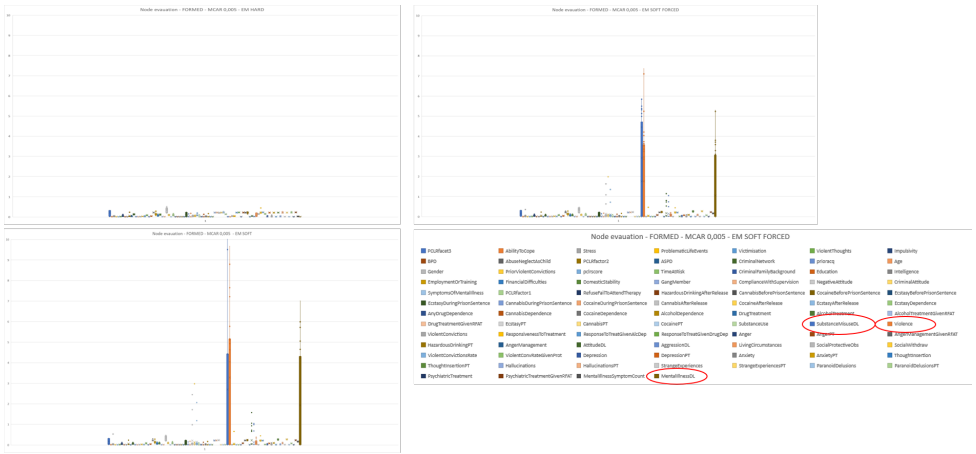


b FORMED - $prop = 0.005$ - KLD- Roots

Figure 4.29. FORMED - $prop = 0.005$ - different types

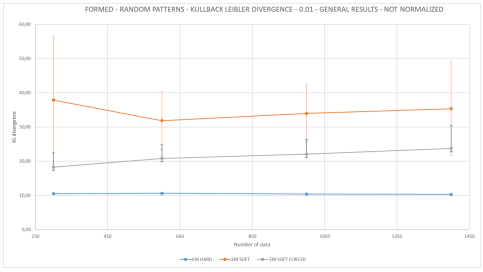


a FORMED - $prop = 0.005$ - KLD- MCAR

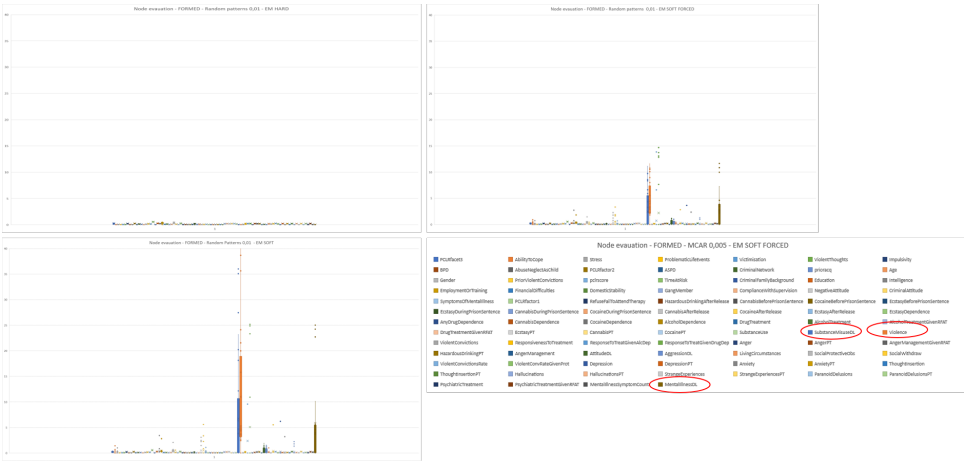


b FORMED - $prop = 0.005$ - KLD- MCAR

Figure 4.30. FORMED - $prop = 0.005$ - MCAR



a FORMED - $prop = 0.01$ - KLD- Random



b FORMED - $prop = 0.01$ - KLD- Random

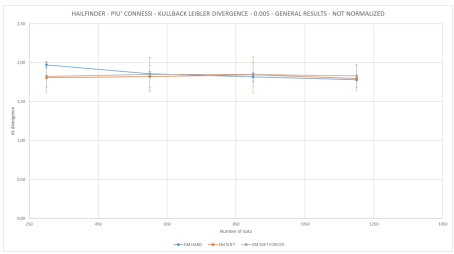
Figure 4.31. FORMED - $prop = 0.01$ - Random Patterns

Discussion of the results

Formed is the first large-size network considered in our experiments. As discussed for the Property network, also in this situation it is necessary to distinguish the different cases carefully. In fact, in some cases, a EM version could be much more efficient than the other versions.
Looking at Figure 4.28 it is possible to observe that all EM versions do not fit very well all the variables. In particular, the divergence of the EM Hard algorithm results higher than EM Soft and EM Forced. This happens due to a variable called `DomesticStability`. However, further experiments have highlighted that EM Hard tends to EM Soft and EM Forced when we increase the number of iterations. Concerning the Formed network, EM Hard makes on average 4 iterations. EM Forced 5 iterations while EM Soft 8 iterations. However, in this case we can conclude that there are not significant differences among the algorithms. Anyway, EM Soft and EM Forced result to be the preferable versions.
Now, we examine the Figure 4.29. This figure summarizes two different experiments based on the more connected nodes and the root nodes. In both cases, as we have seen for leaf nodes, we do not report significant differences among the algorithms. However an important consideration must be done. When we analyze the graph that summarize the computation on the root nodes, it is possible to observe very large confidence intervals. This is an

important result because, when we operate with this network, the initialization and the presence of missing values of the root nodes could greatly influence the results of the EM algorithm. This pattern is not shown in the other graphs.
Until now, no difference among the three algorithms have arisen and the three algorithms have presented excellent results. However, we want to ask what happens when we generate missing values on all variables. As presented with Property, also here we provide two experiment. In the first experiment, missing values are rare in the dataset (Figure 4.30). In the second case, the number of missing values starts to grow and reaches 1% in total (Figure 4.31). It is very interesting to highlight that in this context the same result presented for Property has been achieved. In particular, the lower is the number of missing values the higher is the divergence of the algorithms based on the Soft version. In addition, the divergence increases when we increase the number of iterations. As discussed in the previous Section, also in this case the phenomena can be explained with overfitting. In this case, EM Hard is the best version especially when the frequency of the missing values is sparse and very low into the dataset.

4.5.6 Results with Hailfinder

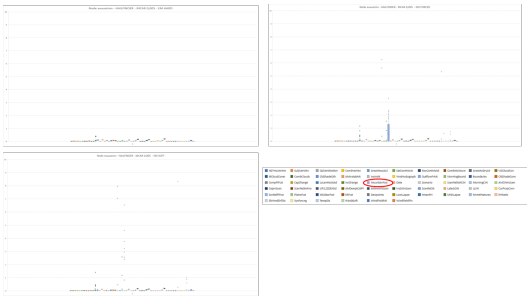


a HAILFINDER - *prop* = 0.005 - KLD- More connected

b HAILFINDER - *prop* = 0.005 - KLD- More connected - Node evaluation

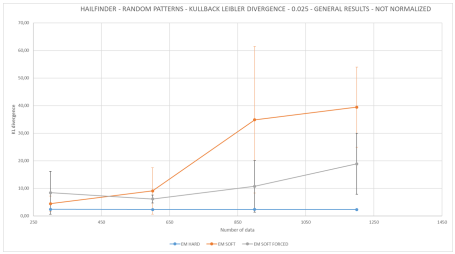
Figure 4.32. HAILFINDER - *prop* = 0.005 - More Connected

a HAILFINDER - *prop* = 0.005 - KLD- MCAR

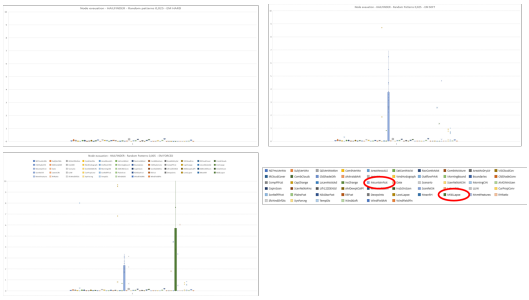


b HAILFINDER - *prop* = 0.005 - KLD- MCAR - Node evaluation

Figure 4.33. HAILFINDER - *prop* = 0.005 - MCAR



a HAILFINDER - *prop* = 0.0025 - KLD- Random Patterns



b HAILFINDER - *prop* = 0.0025 - KLD- Random patterms - Node evaluation

Figure 4.34. HAILFINDER - *prop* = 0.0025 - Random Patterns

Discussion of the results

Hailfinder is the second large-size network that we analyse in our experiments. The particular structure of the Hailfinder network makes its analysis very interesting. When we generate the missing values on a limited number of variables we must distinguish two cases:

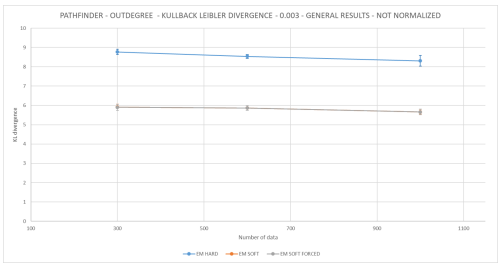
- There is a high number of missing values associated to the node `Scenario` (it is the node with the major outdegree).
- When the analysis does not involve this node because the number of missing values associated to this node is equal to 0 or very low.

In the second case, all algorithms tend to replace missing values in the same way. They do not show statistically significant difference and also the standard deviations and the confidence intervals are very small. However, in the first case, an approach that limits the number of iterations is strongly not recommended. Looking at the Figure 4.32 (b) it is possible to note that EM Forced does not fit the node `Scenario` very well as the other algorithms. In this particular case, when we work with very complex nodes, it is recommended to perform as many iterations as possible. In fact, examining the individual executions carefully, it is possible to highlight that EM Hard stops its execution in 5 iterations (on avarage), EM Soft stops its execution in 9 iterations and EM Forced stops its execution in 4 iterations.
As we saw for other networks, also with Hailfinder, the discussion changes completely when we generate the missing values on all variables in the dataset. In this case we have a sparse distribution of the missing values and EM Soft continues to suffer from the problem of overfitting. This results, well commented before, is shown in the Figures 4.33 and 4.34. Also in this case we present the following results:

- A case in which missing values are generated with the MCAR mechanism and the percentile of the missing values is small.
- A case in which missing values are generated with the MNAR mechanism and the percentile of the missing values is rare.

However, it is important to remark that we report only this two results for simplicity and in order to facilitate the comprehension of results. But, a higher number of experiments have been carried out. In particular, the most important conclusion obtained by the analysis of the results is that EM Soft and EM Forced tend to diverge more when the frequency of missing values is more sparse. In fact, when we work with large datasets, we are going to recommend the use of EM Soft only when the number of missing values is sufficiently high.

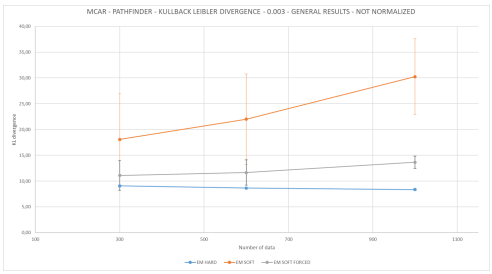
4.5.7 Results with Pathfinder



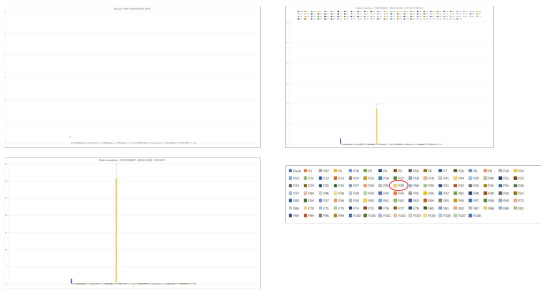
a PATHFINDER - $prop = 0.003$ - KLD- Outdegree

b PATHFINDER - $prop = 0.003$ - KLD- Outdegree - Node evaluation

Figure 4.35. PATHFINDER - $prop = 0.003$ - Outdegree



a PATHFINDER - $prop = 0.003$ - KLD- MCAR



b PATHFINDER - $prop = 0.003$ - KLD- MCAR - Node evaluation

Figure 4.36. PATHFINDER - $prop = 0.003$ - MCAR

Discussion of the results

Pathfinder presents a particular structure whit a single root node that assumes 64 different values. Most of the other nodes are directly linked to this root node called `Fault`. The peculiarity is that most nodes are binaries and, some of them, are strongly unbalanced.
Looking at the results, in general, there is not differences between EM Hard and EM Soft. Anyway, there are strong differences when we generate missing values on the root node

`Fault`. This is the case reported in the Figure 4.35. Of all the cases analyzed in our experiments, this one is the most obvious. When we work with this types of nodes, EM Hard result to be strongly not recommended because it is not able to consider all possible cases of completion of the variables. The result is that EM Hard diverges more than the other algorithms. Also by increasing the number of iterations, the result remains unchanged. Through the node evaluation displayed in Figure 4.35 (b) it is possible to highlight very well that EM Hard fails to fit the node `Fault`. On the contrary, EM Soft and EM Forced present excellent performance. This is an important results which must be taken into account when we must choose the algorithm to be applied on the partial dataset.
Looking at the Figure 4.36, another case similar to the previous ones is exposed. Also in this case, EM Soft and EM Forced tend to overfit.

This discussion concludes the presentation of the results. In the next Chapter we are going to present the conclusions of this work with the aim to summarize the most important insights and provide useful information in order to select the algorithm to apply.

Chapter 5

Conclusions

In the previous Chapter we have presented and analysed the EM algorithm. We highlighted that it is a powerful tool and it is able to work on every type of dataset. We distinguished three types of versions: EM Hard, EM Soft and EM Forced and we emphasised that the study of these three versions could be very stimulating because, in the literature no implementation of the EM Soft version was presented. The analysis of the results was an amazing moment of satisfaction for us. Before proceeding with the presentation of the insights obtained from results analysis, I would like to express my thanks to the people who have taken part in this work since January 2020. Thanks to my Professor Fabio Stella who followed us, made himself available to us everyday and gave us very useful indications. Thanks to Ph.D. Marco Scutari, founder of the `bnlearn` library on R, who has always been available to participate to our presentations. Many thanks to Francesco Stranieri, my close working partner. We studied together and we asked for advice in the most difficult moments, it was a satisfaction to carry on other projects. Thanks to the whole MADLAB team and in particular to Ph.D. Alessandro Bregoli my correlator. Thanks to Alessio and Emanuele who have always worked in parallel in another wonderful project but they have always been present at our presentations.
Anyway, our discussion regarding the EM algorithm is not finished because we have to treat the third goal: Providing useful tips for the people who have to deal with missing datasets. More precisely, in the previous Chapter we reported the most important results but we still have not provided the most useful indications about which algorithm should be selected. The first consideration when we deal with EM algorithm is that it is not easy to determine in advance which version of the Expectation-Maximisation algorithm is the best. However, after experimenting and validating the results on several datasets we have discovered some very interesting insights. The choice mainly depend on the properties of the dataset and it is possible to summarize the choice of the best algorithm through the tree structure reporting in the Figure 5.1.

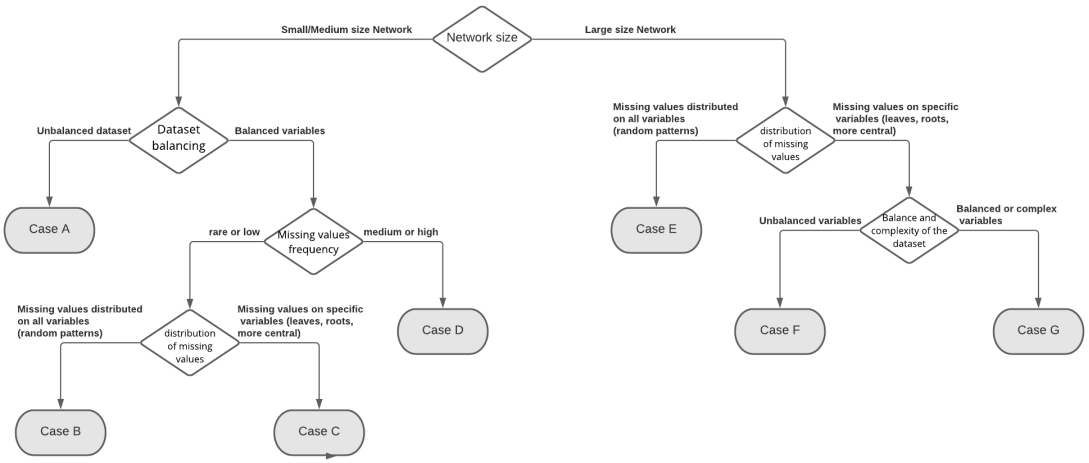


Figure 5.1. Decision tree for selecting the best algorithm

It is also possible observe the following tree in the Appendix B in a larger format. From this tree it is possible differentiate different cases that could be summarize in the Table 5.1.

Case	**Best Algorithm**	**Ideal number of iterations**	**Experimented Dataset**
A	EM Hard EM Soft/forced	[3,4]	ASIA ALARM
B	EM Hard	[2,3]	SPORTS PROPERTY
C	EM Soft/forced	[3,6]	SPORTS PROPERTY
D	EM Hard	[3,6]	SPORTS PROPERTY
E	EM Hard	[3, max]	FORMED PATHFINDER HAILFINDER
F	EM Hard	[3, 6]	FORMED PATHFINDER HAILFINDER
G	EM Soft	[5, max]	FORMED PATHFINDER

Table 5.1. Results observed from the analysis of the results

Where *max* is related to the maximum number of iterations defined for the execution of the EM algorithm. Now, we are able to go into the details of the specified cases. **Case**

A describes a classic case where the variables are strongly unbalanced (we refer to the Section 4.1) and the network is very simple. An unbalanced dataset proves to be easier to learn than a strong balanced dataset because the learning algorithm learns the most probable values better and faster. The only risk that is important to avoid is the overfitting phenomenon. This risk occurs when the probability distribution of the partial variables is learned too well from the learning algorithm. This risk increases when we have to do with large-size networks as described in the experiments. However, with reference to the case presented, the increase in the number of iterations and the number of missing values does not produce statistically significant differences and all the EM versions try to learn exactly the same probability distribution. Since there are no significant differences, we recommend to use both EM Hard and EM Soft. Moreover, we recommend to set a low number of iterations especially if we deal with small-size networks.
Looking at the **case B**, a very particular case is highlighted. In this situation the missing values are sparse on the dataset and they show a low frequency. It is rare to find missing values and the EM algorithms learn very well the distribution of probabilities of each variable from the first iterations. In this situation, it is very important to set a low number of iterations in order to avoid overfitting. Usually, two iterations could be enough. As far as the algorithm to select is concerned, we strongly recommend EM Hard because proves to have a stable trend with more pronounced standard deviation. More precisely, EM Hard tends to overfit slower than EM Soft. As the data increase, the divergence of EM Soft is higher.
Now, we focus our attention on the **case C**. In this circumstance, the frequency of missing values is low but the missing values are concentrated only on specific variables as the leaf nodes. When the missing values are concentrated only on the target attribute, we could see the application of the EM algorithm as a problem of classification. The peculiarity of this situation is that EM tries to learn very well all the probability distribution of all the variables but a noise is applied in the learning of the partial observed variables. This noise could be propagated on the network and the overfitting phenomena (unlike case B) are more difficult. On the contrary, if it is true that finding overfitting phenomena is difficult, it is also true that the performance is lower than the previous case. Performance is lower because when we deal with strongly balanced variables, we propagate uncertainty in the network and sometimes the algorithms are not able to make the best decision with a high degree of certainty. For this reason the best algorithm is EM Soft and we recommend to fit it at most 5 iterations. In fact, when we work with a small/medium networks with a limited number of parameters, the convergence of the EM algorithm is very fast.
Case D describes a very interesting case. In fact, this is a classic situation in which the dataset shows a high number of missing values (for example 10%). In this case the variables are strongly balanced but a strong noise is present due to missing information. In this situation, the presence of missing values could significantly deviate the probability distribution of a variable from the uniform distribution. It is observed that when EM Hard fails to fit a variable also EM Soft fails. In particular, the divergence of the two EM algorithms is very similar and, it is possible to provide a very useful indication. In the situation just presented, the number of missing values tend to be high. Because EM is a computationally intensive algorithm, it is better to choose a lighter version. For this reason, we recommend to choose the EM Hard algorithm. This discussion concludes the cases relating to the small/medium networks.

Now, we present the **Case E**. This situation represents the very particular case which the distribution of missing values is uniform for all variables. All variables have a probability distribution that differs from the real distribution. Therefore, each variable is characterized by a specific noise that propagates within the network. If we learn very well the probability distribution of the partial observed variables, the risk is to generate overfitting phenomena. In this situation EM Hard tends to have a stable trend with a more pronounced standard deviation. On the contrary, EM Soft tends to show an increasing trend in terms of Kullback-Leibler divergence as the number of iterations increase. In this case we recommend EM Hard and we recommend to carry out experiments by varying the number iterations and we advice to save the partial results. In fact, when we apply EM Hard on large-size networks we report two different types of results. Sometimes the best performance is obtained with a low number of iterations. In other cases, performance increases up to a maximum point when we increase the number of iterations.
Let us now illustrate **case F**. In this situation, the missing values are concentrated in some variables and the values assumed by these variables tend to be unbalanced. The peculiarity is that EM Hard and EM Soft proves to show the same results without showing statistically significant differences. However, we recommend EM Hard because it shows a smaller variance. In addition, EM Hard is preferable for the computation time.
The final case is **Case G**. It is a very particular situation which needs to be treated carefully. In this case, the missing values are concentrated on a subset of variables (as in Case F). However, in this situation, many of the partial observed variables are balanced or they exhibit a high number of values. In this case, an algorithm as EM Hard is not recommended because it selects only one of the possible assignments without considering all possible cases of completion. This fact introduces a non-ignorable bias and also the performance of EM Hard results to be lower than EM Soft. In particular, through a node-by-node analysis, we reported that EM Hard does not fit very well in these variables and the error could propagate within the network. It is possible to conclude that, in this case, EM Soft is recommended because it is able to minimize this error.

These cases are obtained from the analysis of the results and they represent an important advice to the reader who intends to use the EM algorithm to replace missing values within a dataset . However, further considerations have been observed during our analysis:

- EM Hard converges faster and with a lower number of iterations. On the contrary, EM Soft could converge very slowly because at each iteration, it changes the parameters more cautiously than the version EM Hard. In particular, many experiments have highlighted that EM Hard tends to converge on the third iteration. EM Soft could converge from the sixth iteration.

- The choice of the parameter `Number_iteration` is fundamental in EM Soft. A very high number of this parameter could lead to overfitting. Vice versa, very low values, may not be enough to fit the data effectively. In EM Hard the choice of the number of iterations is important, but it has been observed a less significant difference in terms of divergence than EM Soft.

- In general if we look at the confidence intervals, it is possible to highlight that EM Hard provides more stable results with a smaller standard deviation than EM Soft.

Whenever it is not easy to determine which algorithm to apply, the recommended choice is to prefer the EM Hard version.

- In addition, EM Hard is also recommended when the hardware capabilities are limited. When the computational time is a relevant problem is preferable to to choose a faster algorithm.
- In our discussion we have not mentioned at the different types of missing values. In fact, in general, when we work with dataset that present missing values generated completely at random (MCAR mechanism), the same considerations apply as for the MNAR and MAR data are valid.

These points conclude the considerations regarding the EM algorithm. However, many aspects can still be improved and we are going to discuss these aspects in the next Chapter concerning the future developments.

Chapter 6

Future developments

Despite the fact that EM algorithm works very well on all types of network, different improvements can still be made. In fact, we implemented the EM algorithm with the final goal to provide a package of methods that made available the EM algorithm both in Hard and Soft version. This package works only on discrete data (and with discrete Bayesian networks) and it is not possible to use the EM algorithm when data are continuous. However, we report a list of future developments that are very useful to implement.

Optimization of the EM algorithm

We have discussed how the optimization of the methods could lead to an improvement in terms of computational time. During the implementation, we have just introduced optimisation methods based on the application of the R best practices and the introduction of the parallelizzation. However, further improvements can be made. For example:

- Using the approximate inference whenever the application of the exact inference is very expensive.
- Making the data structures more accessible, allowing the punctual access to the values.
- parallelizing the execution inference algorithm.
- Reducing the memory usage through the redefinition of data structures introduced in this thesis.

Comparison between EM algorithm and the other algorithms

We have discussed that the results using the soft version of EM algorithm are very poor because nobody has never implemented a full version of the EM algorithm. For this reason, it is not possible to establish in advance which algorithm must be chosen to solve the problem of the replacement of missing values. The results in the literature are more complete when we compare the EM algorithm with other category of algorithms as the single imputation methods or the Multiple Imputation algorithm. Fortunately, my colleague Francesco Stranieri has already started this difficult task by comparing the EM algorithm with the with very consolidated algorithms such as: Mice, K-nearest neighbors and Random Forests.

He has experimented with two different datasets: Asia and Alarm and very interesting results have been obtained. Continue in this direction is crucial in order to determine the contexts in which the application of the EM algorithm results to be the best.

Adapt the EM algorithm to work with string values

The main limitation of the actual version of the EM algorithm is that the strings have to be converted in numerical values through the definition of a dictionary. If the dataset contains string values, EM converts them into numbers automatically by applying the `as.numeric` function on the dataset. The use of numeric values has simplified the execution of the exact inference and methodological approach. For examples, if a binary attribute can assume the values *"no"* and *"yes"*, EM converts these two strings with 0 and 1 respectively. The same discussion is valid when the values are *"false"* and *"true"*. On the contrary, if a attribute can assume the two different values (e.g. *"bad"* and *"good"*) or more than two values, for example: *"insufficient"*, *"sufficient"*, *"good"* and *"excellent"*, EM sorts the string values in ascending alphabetical and it assigns the numbers: *excellent: 0*, *good: 1*, *insufficient: 2*, *sufficient: 3*. Unfortunately, the actual output of EM is a numeric dataset. A very useful future development is to create a method that restores the previous string values.
Anyway, a second motivation which led to work with numerical datasets is that we have used the `ampute` method in order to generate the missing values. The peculiarity of this method is that it maps all strings in numerical representation.

EM algorithm on a larger number of datasets

We have experimented our implementation of EM algorithm with 7 very different types of network. These networks exhibit very different properties both in terms of size, complexity and probability distribution. With regard to the size, we distinguished three different types of networks based on the number of nodes: small-size networks, medium-size networks and large-size networks. With regard to the complexity, we made considerations both in terms of network complexity (maximum indegree, outdegree, number of arcs, density of the networks etc.) and in terms of complexity of the variables (number of values that each variable can assume). With regard to the probability distribution, we have distinguished the cases in which we have available balanced datasets from the cases in which strongly unbalanced datasets are available.
It is possible to extend our study by searching other networks which exhibit different properties. The observations of new insights is very useful when we have to recommend a version to a user. By using of a decision tree like the one shown in the figure 5.1, a user will be able to identify the best algorithm which can be used immediately.

Integrate these methods into a library

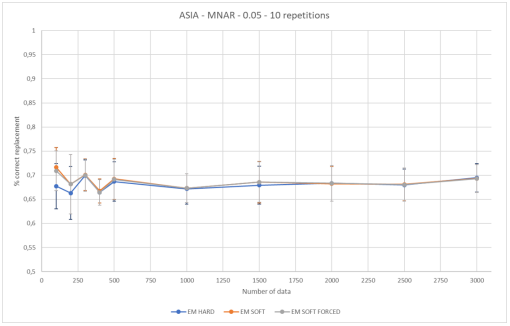
The use of EM algorithm is very simple. After loading all the methods, it is sufficient to call the functions `em_hard` or `em_soft` to execute the EM algorithm. The source code is open source available on GitHub. However, integrating these methods into a library is a step that can facilitate the use of the EM algorithm.

Appendix A

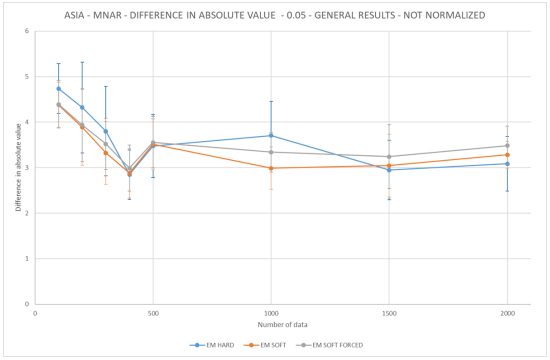
Appendix - All results

ASIA

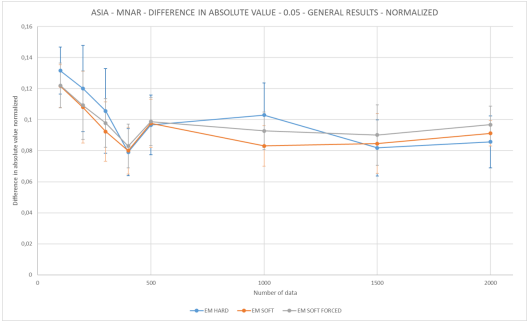
Random patterns - prop = 0.05



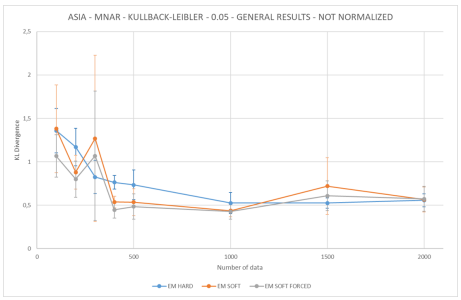
a



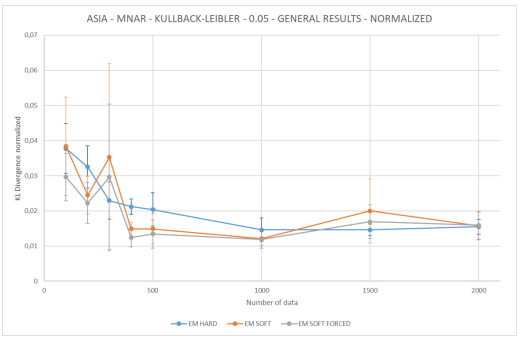
b



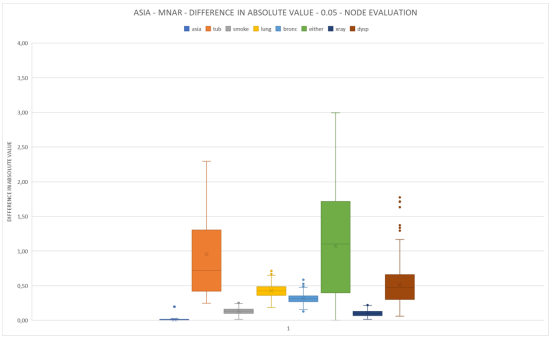
c



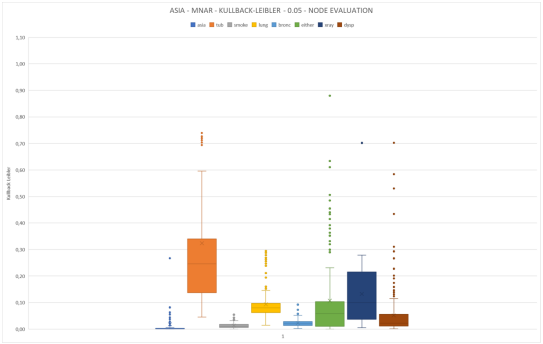
d



e

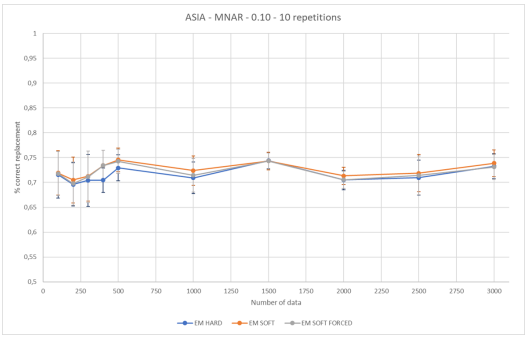


f

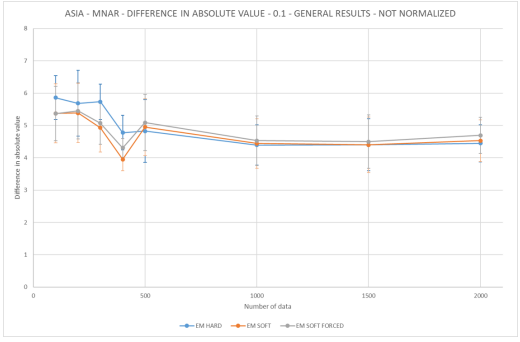


g

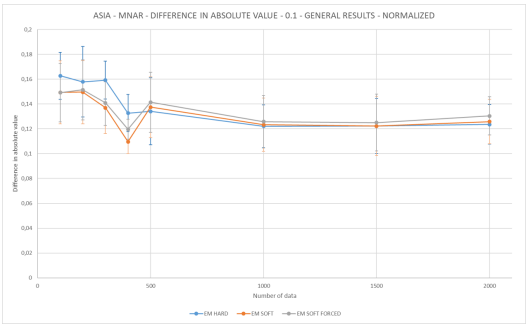
Random patterns - prop = 0.1



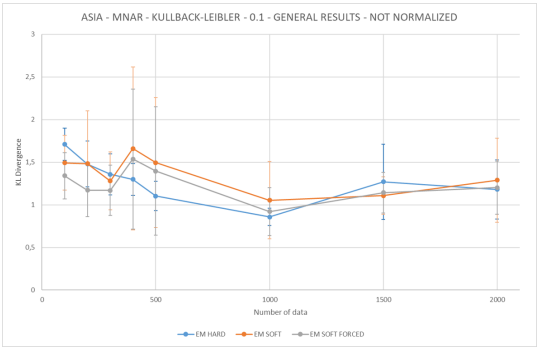
h



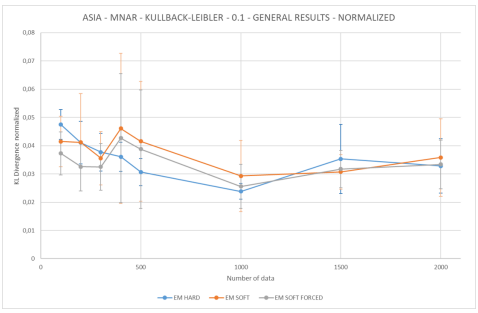
i


ASIA - MNAR - DIFFERENCE IN ABSOLUTE VALUE - 0.1 - GENERAL RESULTS - NORMALIZED
Difference in absolute value
Number of data
EM HARD
EM SOFT
EM SOFT FORCED

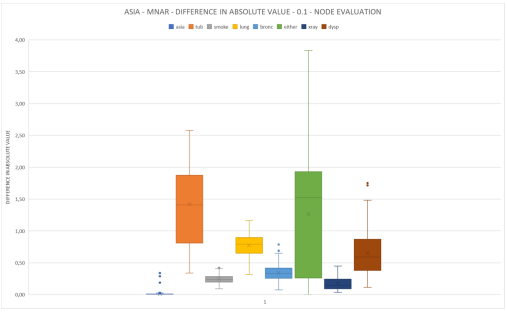
j


ASIA - MNAR - KULLBACK-LEIBLER - 0.1 - GENERAL RESULTS - NOT NORMALIZED
KL Divergence
Number of data
EM HARD
EM SOFT
EM SOFT FORCED

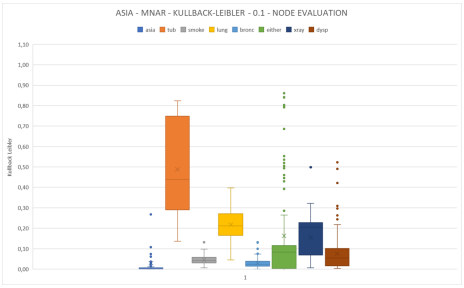
k


ASIA - MNAR - KULLBACK-LEIBLER - 0.1 - GENERAL RESULTS - NORMALIZED
KL Divergence normalized
Number of data
EM HARD
EM SOFT
EM SOFT FORCED

l

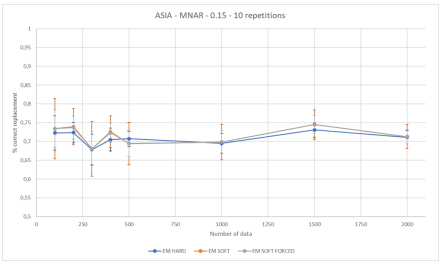

ASIA - MNAR - DIFFERENCE IN ABSOLUTE VALUE - 0.1 - NODE EVALUATION
asia
tub
smoke
lung
bronc
either
xray
dysp

m

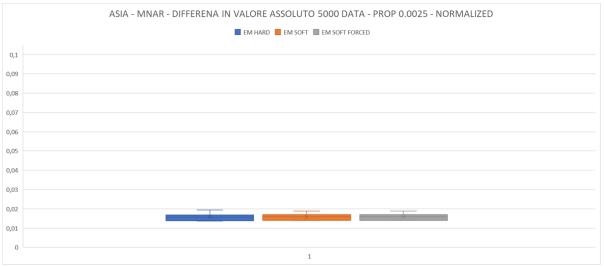

ASIA - MNAR - KULLBACK-LEIBLER - 0.1 - NODE EVALUATION
asia
tub
smoke
lung
bronc
either
xray
dysp

n

Random patterns - prop = 0.15

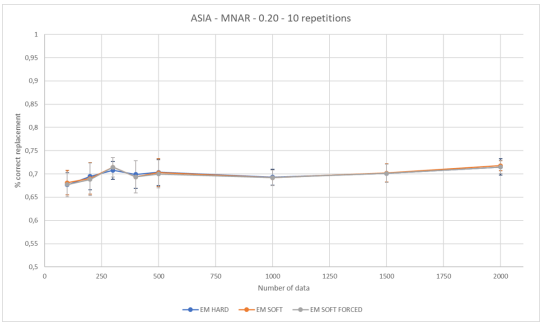


o

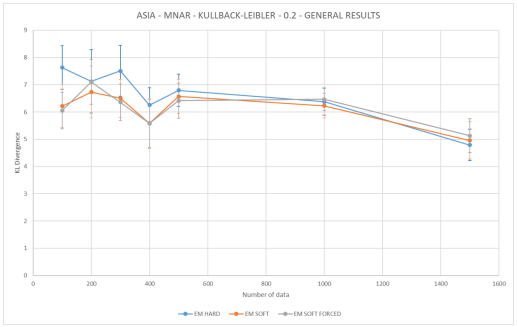


p

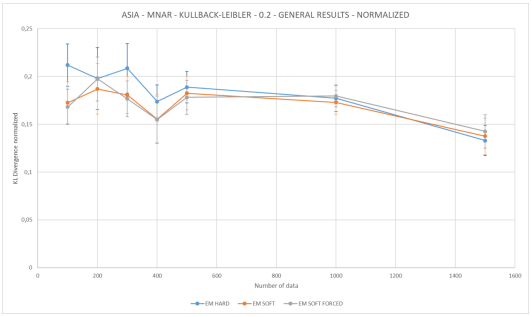
Random patterns - prop = 0.2



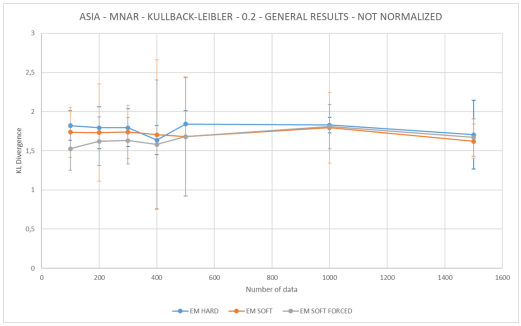
q



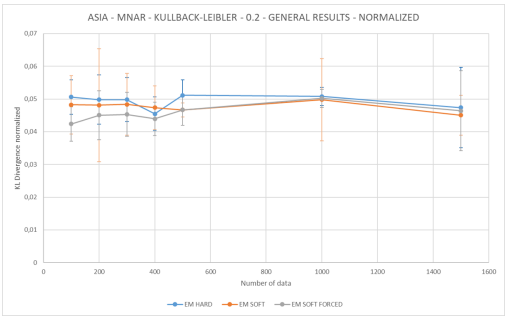
r



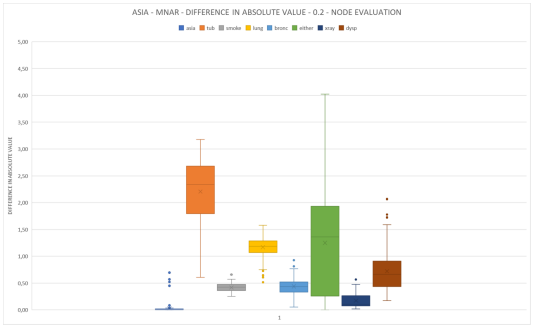
s



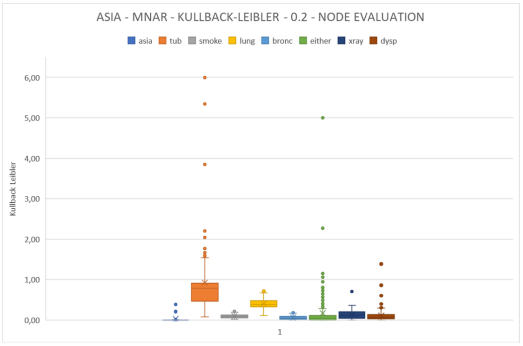
t



u

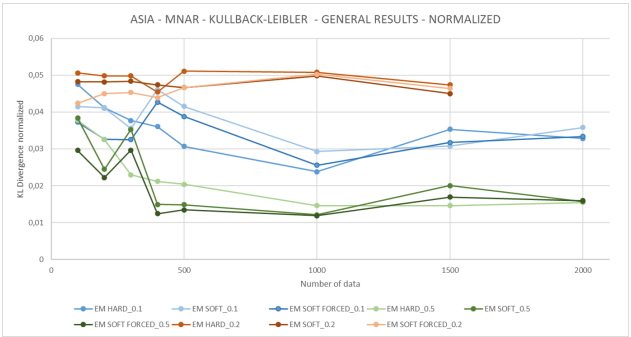


v



w

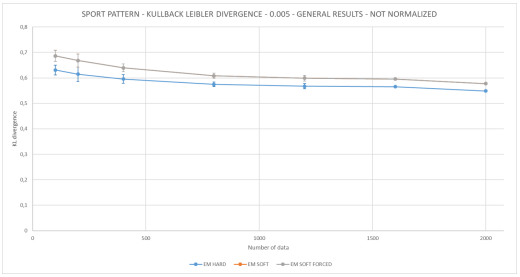
ASIA - prop comparison



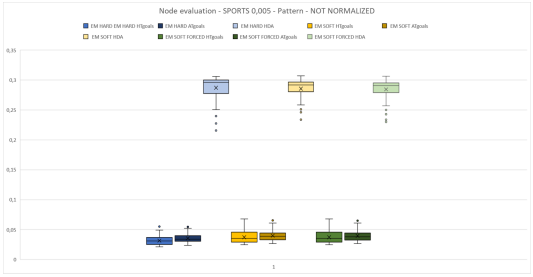
x

SPORTS

Most central nodes - prop = 0.05

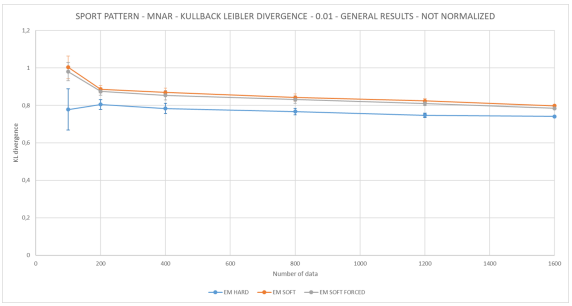


y

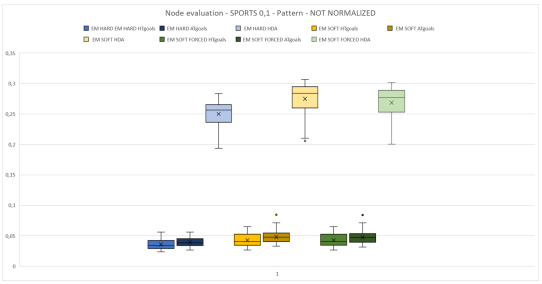


z

Most central nodes - prop = 0.1

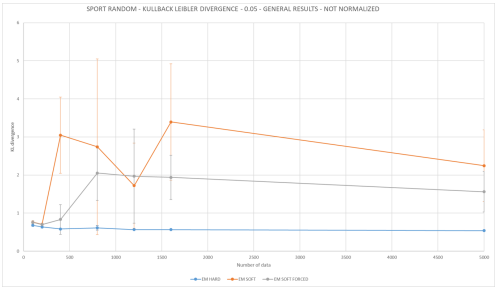


aa

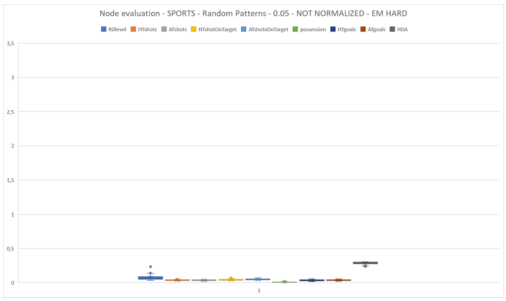


ab

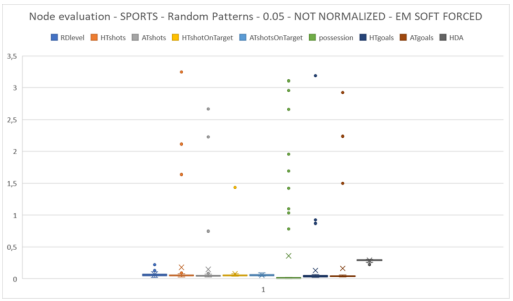
Random patterns - prop = 0.05



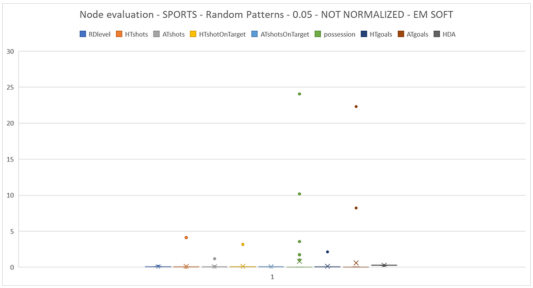
ac



ad

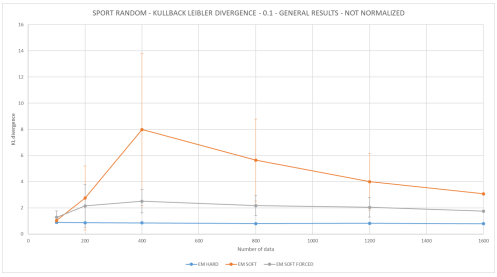


ae

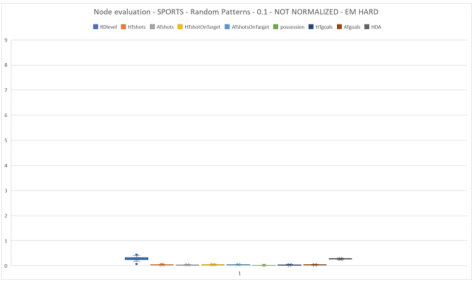


af

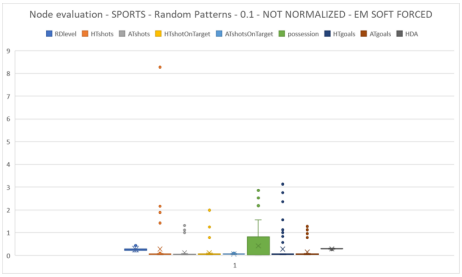
Random patterns - prop = 0.1



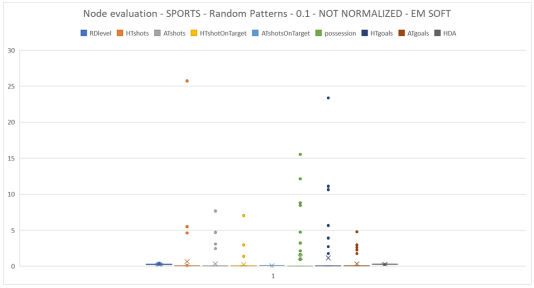
ag



ah



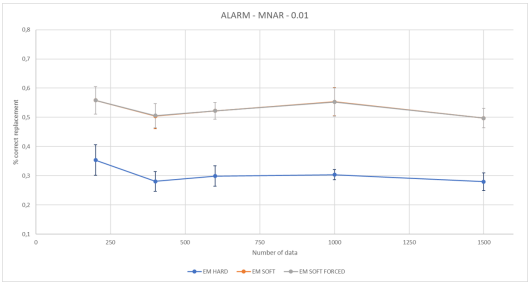
ai



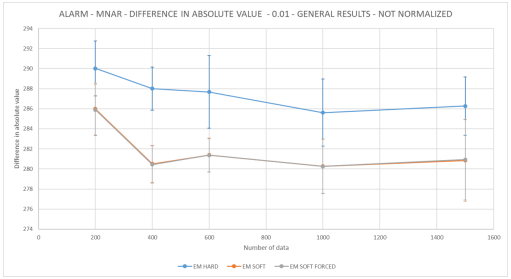
aj

ALARM

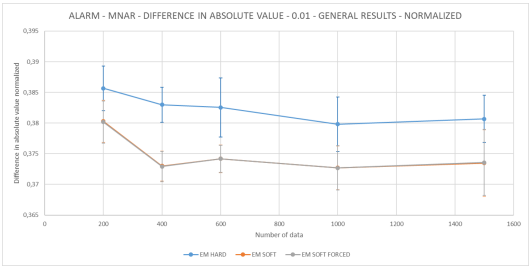
Random patterns - prop = 0.01



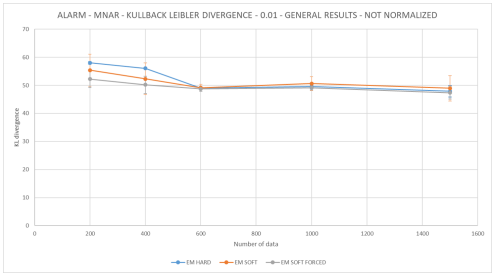
ak



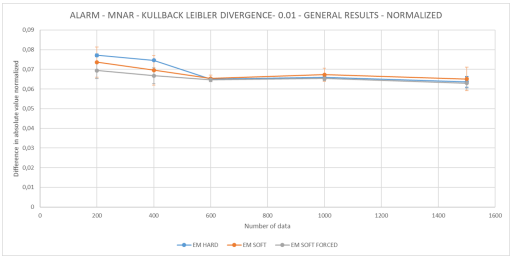
al



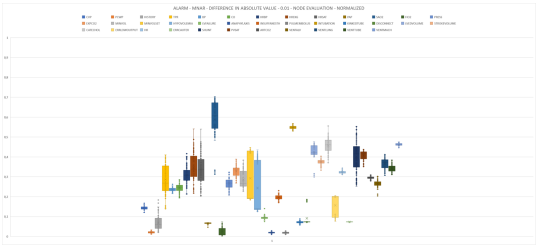
am



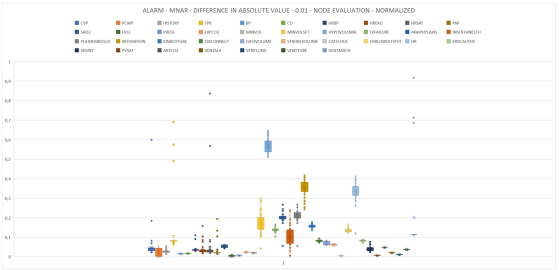
an



ao

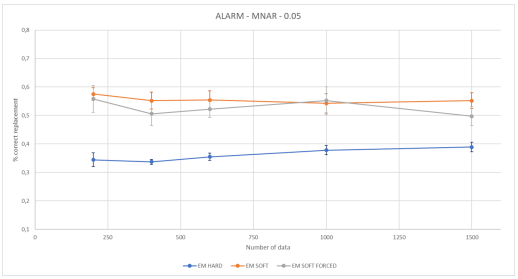


ap

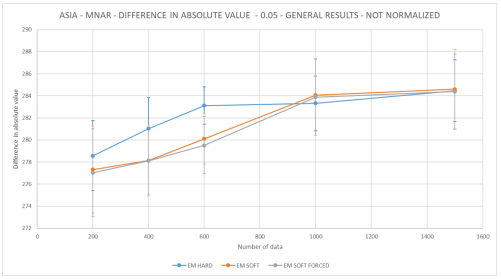


aq

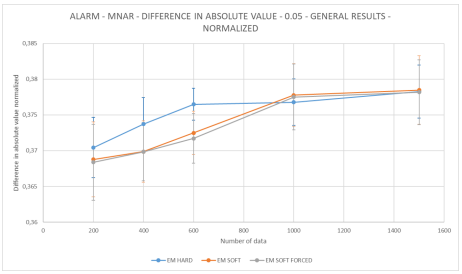
Random patterns - prop = 0.05



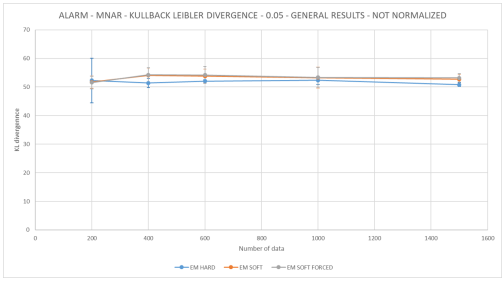
ar



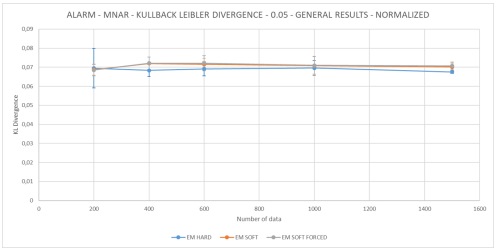
as



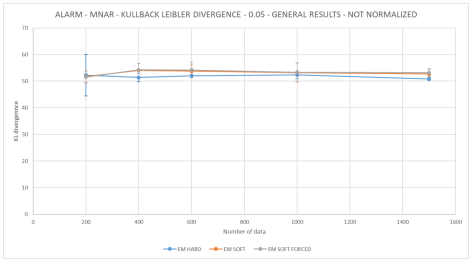
at



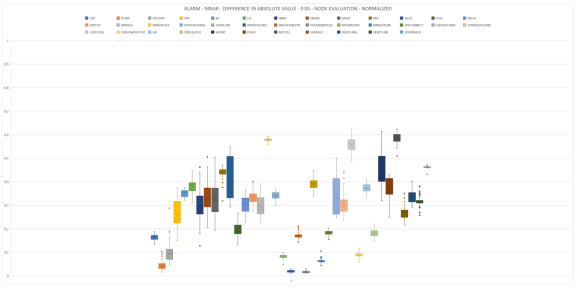
au



av



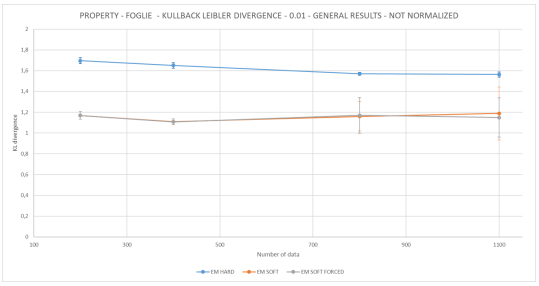
aw



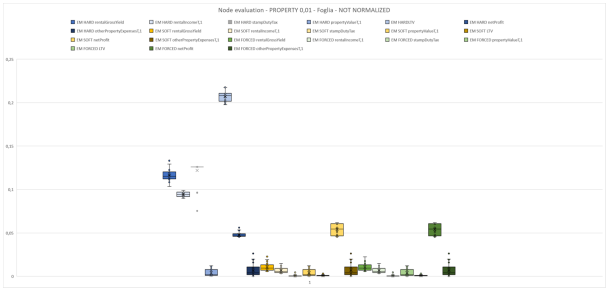
ax

PROPERTY

Leaf nodes - prop = 0.01

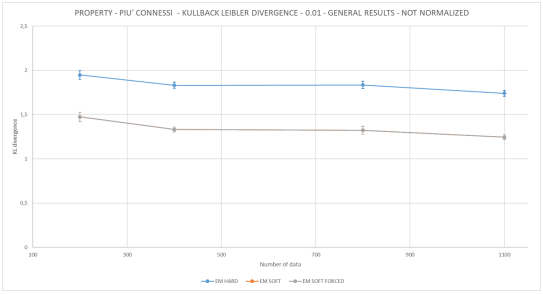


ay

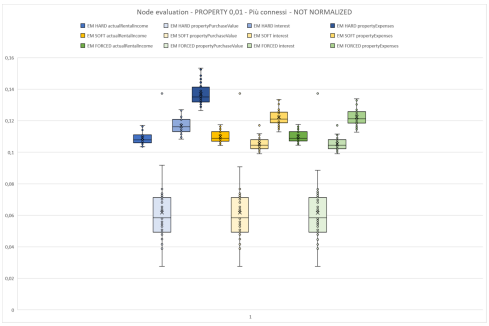


az

Most connected nodes - prop = 0.01

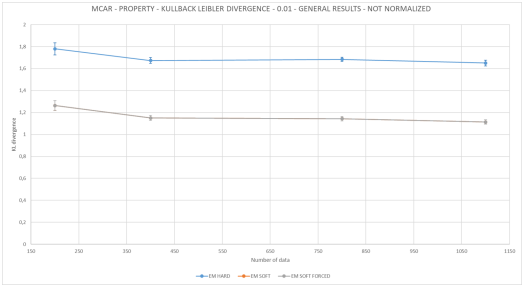


ba

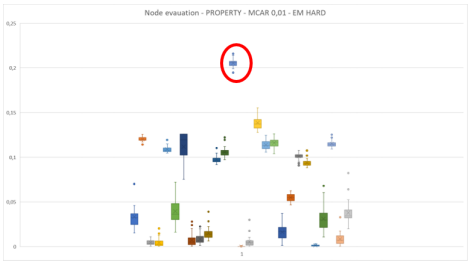


bb

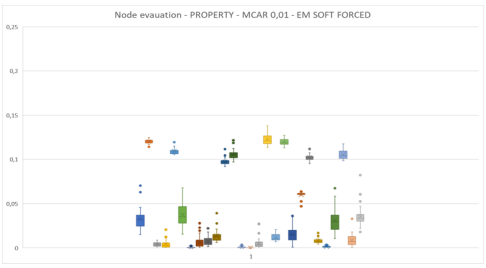
Random patterns with MCAR mechanism - prop = 0.01



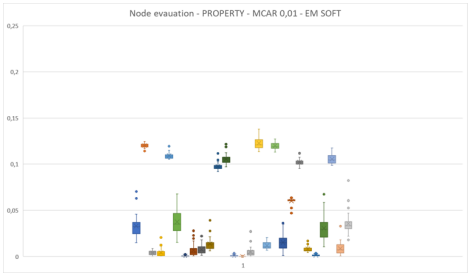
bc



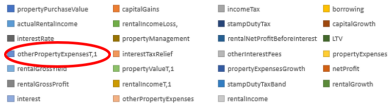
bd



be



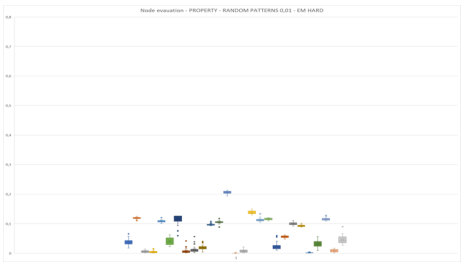
bf



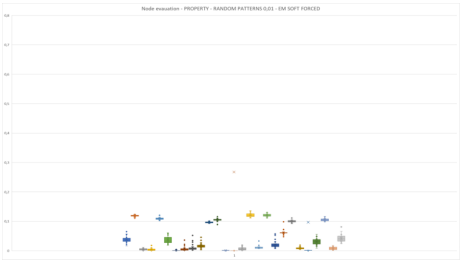
bg

Random patterns with MNAR mechanism - prop = 0.01

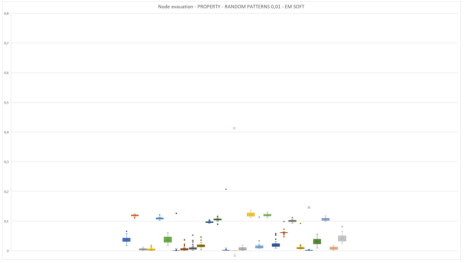
bh



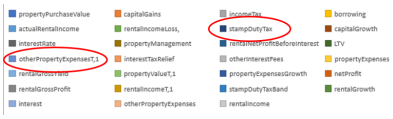
bi



bj

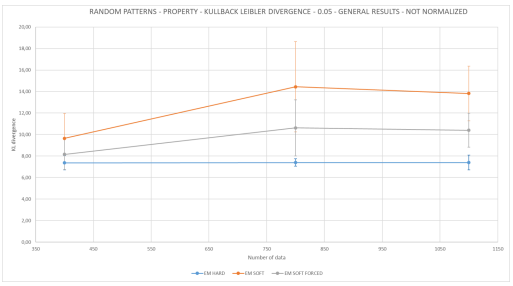


bk

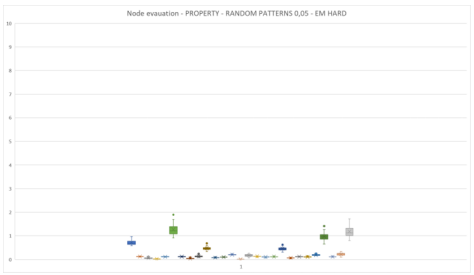


bl

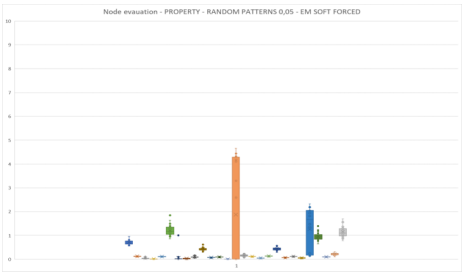
Random patterns with MNAR mechanism - prop = 0.05



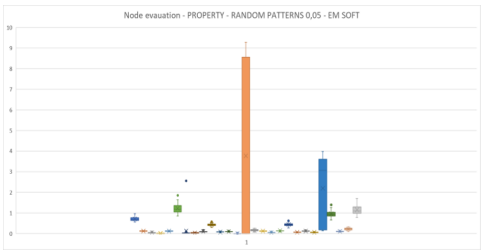
bm



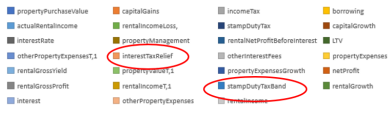
bn



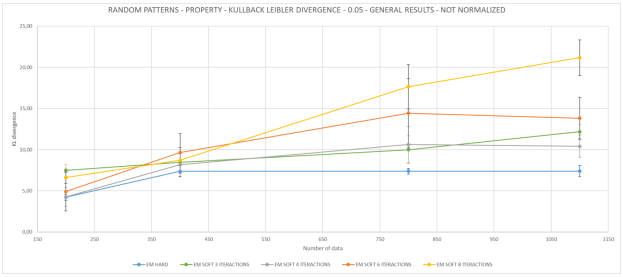
bo



bp



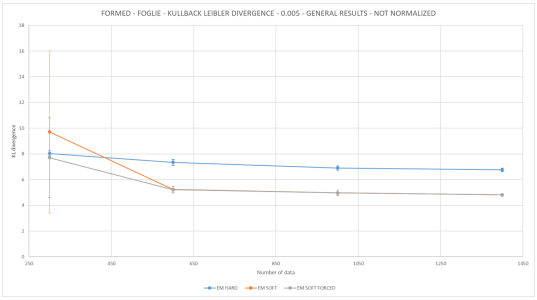
bq



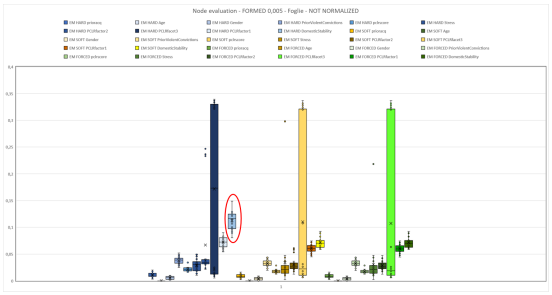
br

FORMED

Leaf nodes - prop = 0.005

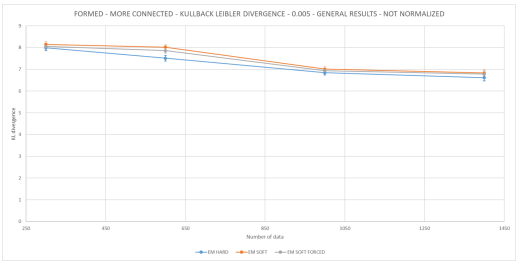


bs

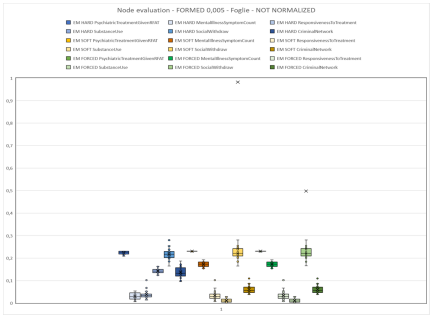


bt

Most connected nodes - prop = 0.005

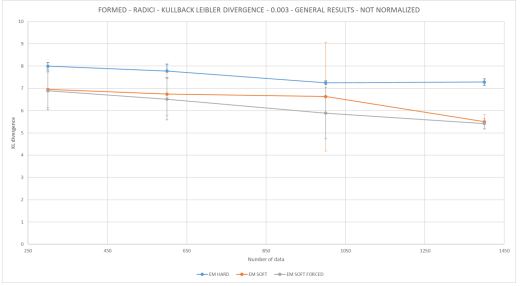


bu



bv

Root nodes - prop = 0.003



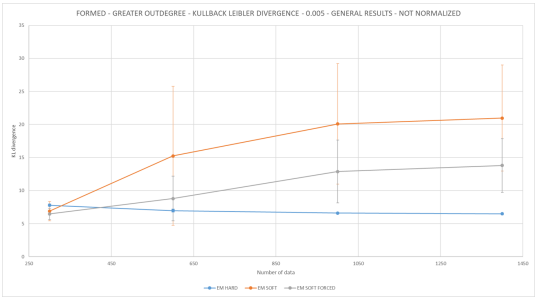
bw

bx

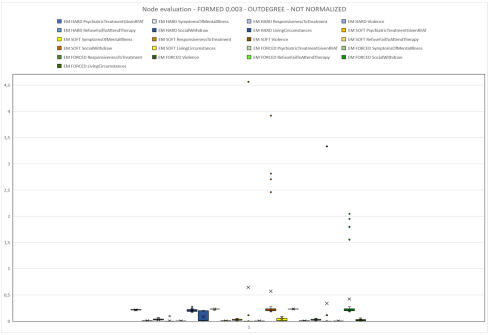
by

bz

Nodes with greater outdegree - prop = 0.003

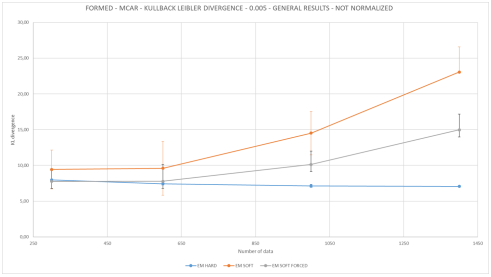


ca



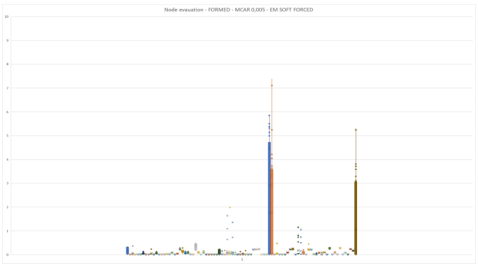
cb

Random patterns with MCAR mechanism - prop = 0.005

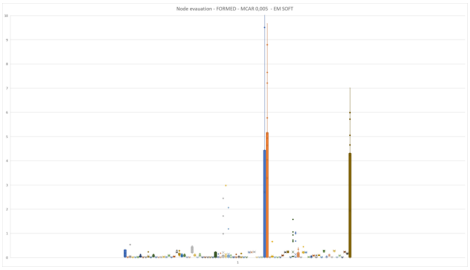


cc

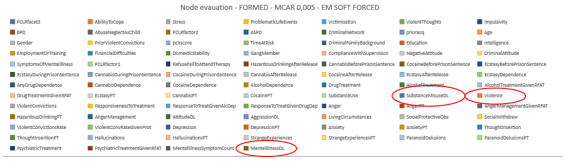
cd



ce

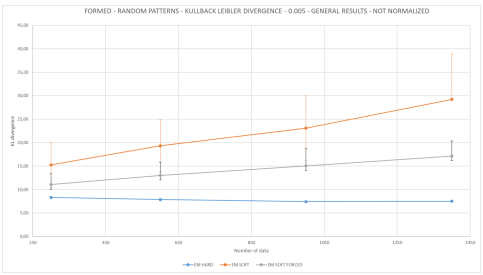


cf



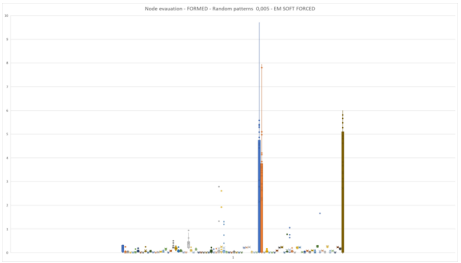
cg

Random patterns with MNAR mechanism - prop = 0.005

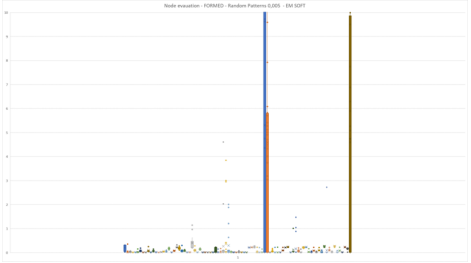


ch

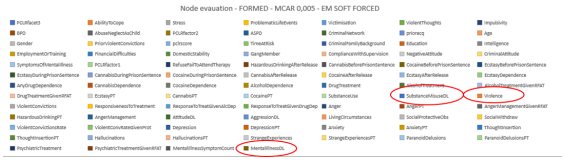
ci



cj

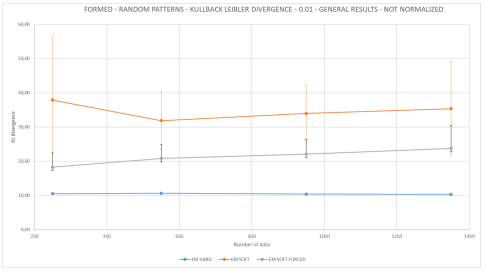


ck



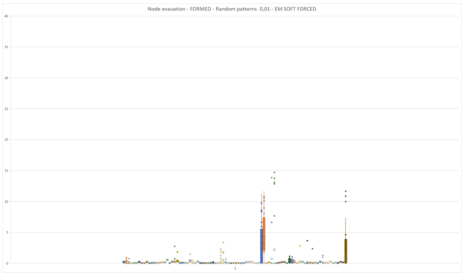
cl

Random patterns with MNAR mechanism - prop = 0.01

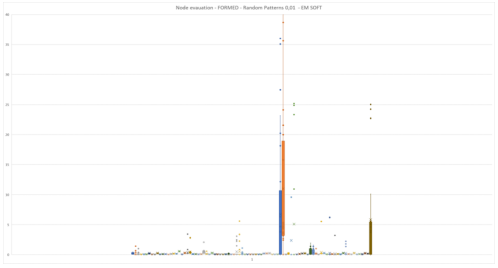


cm

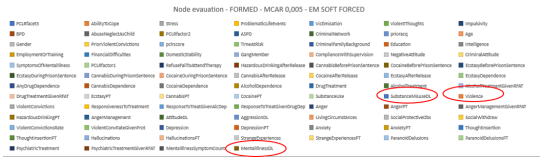
cn



co



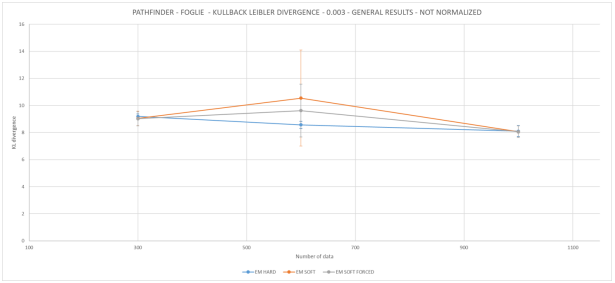
cp



cq

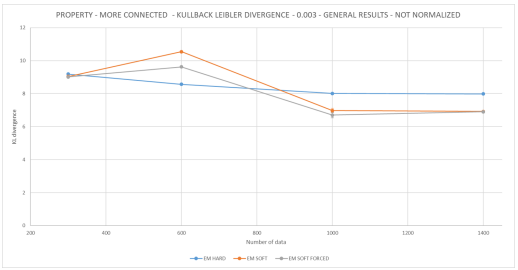
PATHFINDER

Leaf nodes - prop = 0.003

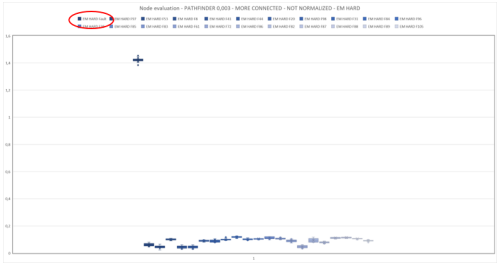


cr

Most connected nodes - prop = 0.003

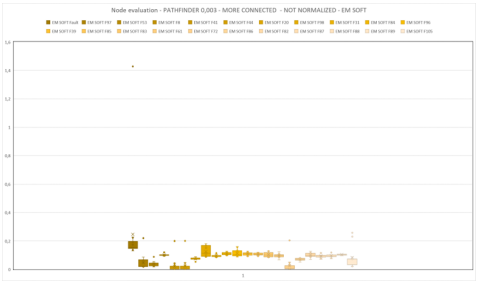


cs



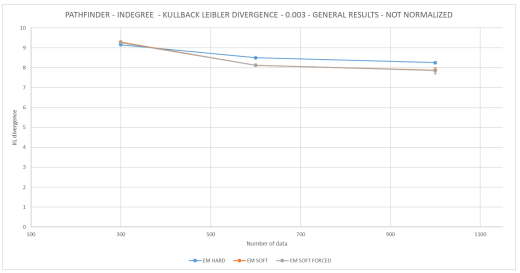
ct

cu

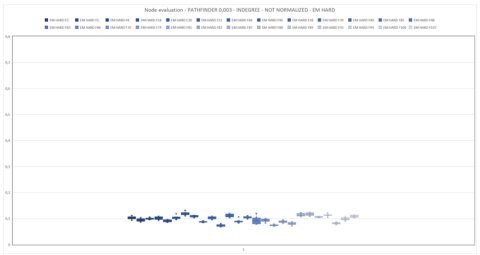


cv

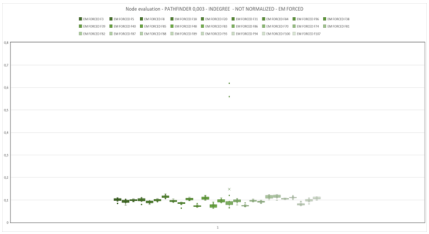
Nodes with greater indegree - prop = 0.003



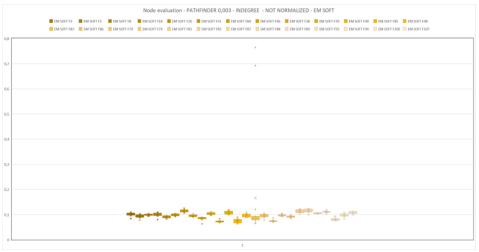
cw



cx

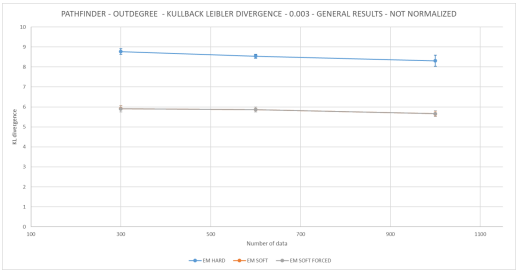


cy

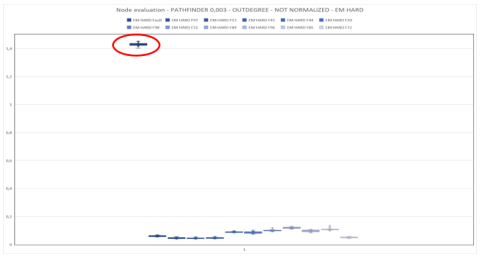


cz

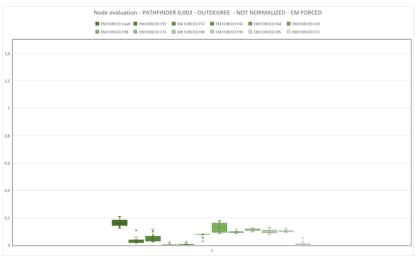
Nodes with greater outdegree - prop = 0.003



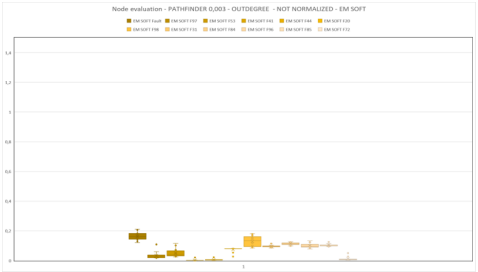
da



db



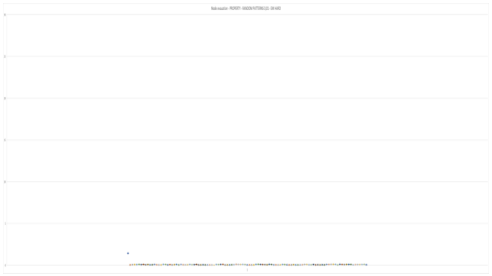
dc



dd

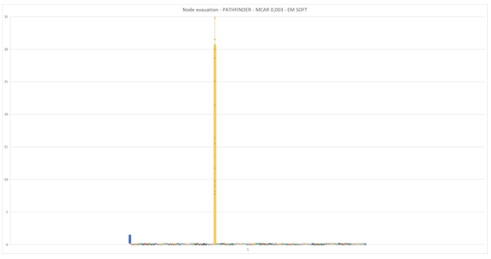
Random patterns with MCAR mechanism - prop = 0.005

de



df

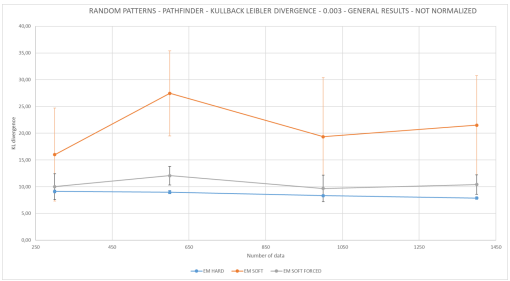
dg



dh

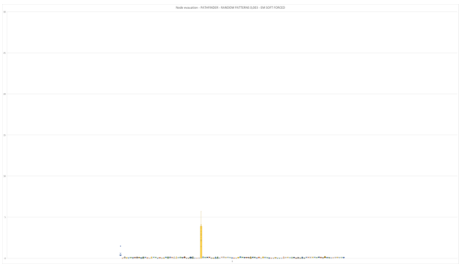
di

Random patterns with MNAR mechanism - prop = 0.003

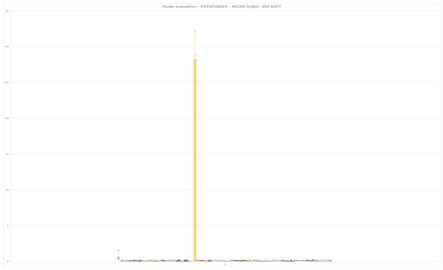


dj

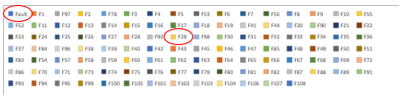
dk



dl

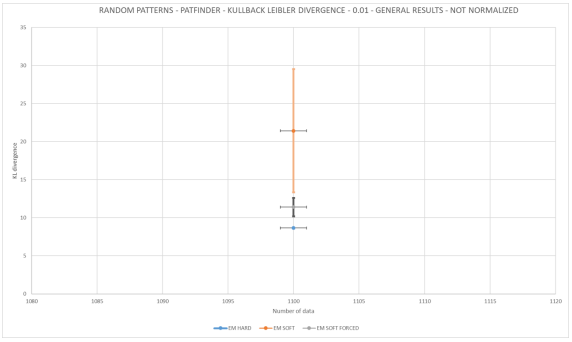


dm



dn

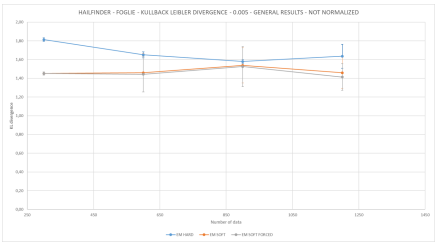
Random patterns with MNAR mechanism - prop = 0.01



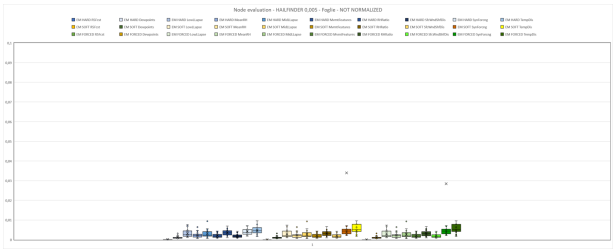
do

HAILFINDER

Leaf nodes - prop = 0.005

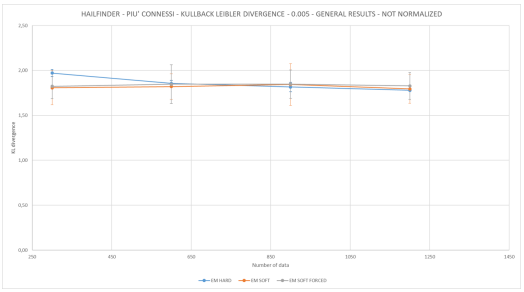


dp

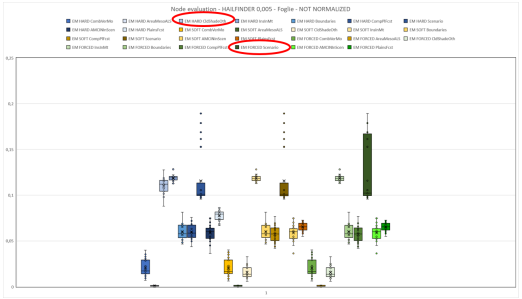


dq

Most connected nodes - prop = 0.005

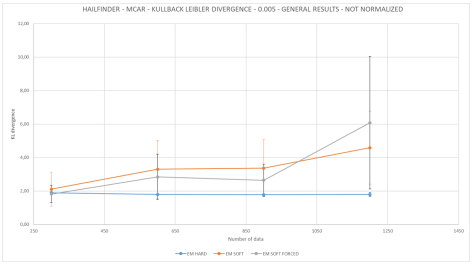


dr

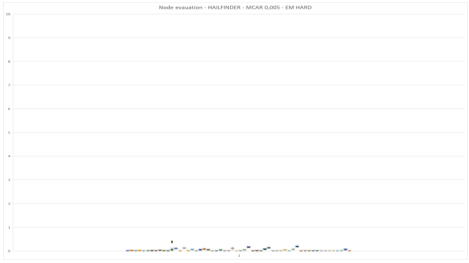


ds

Random patterns with MCAR mechanism - prop = 0.005



dt



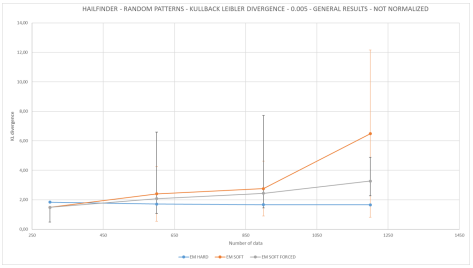
du

dv

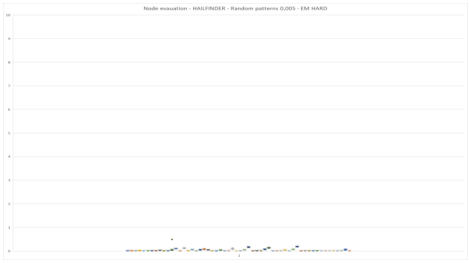
dw

dx

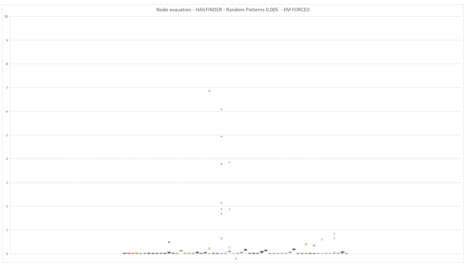
Random patterns with MNAR mechanism - prop = 0.005



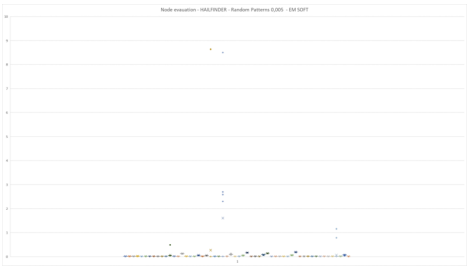
dy



dz



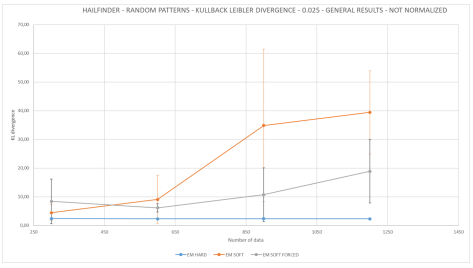
ea



eb

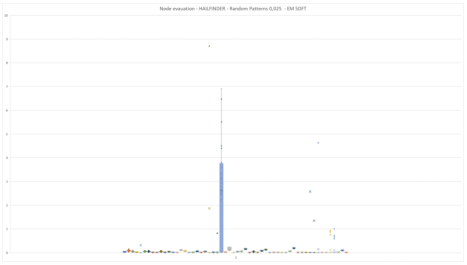
ec

Random patterns with MNAR mechanism - prop = 0.025

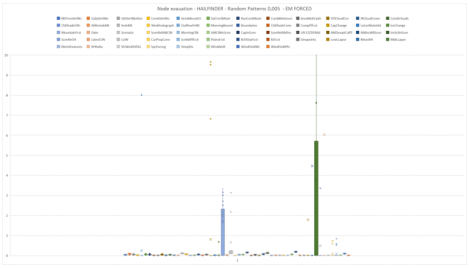


ed

ee



ef



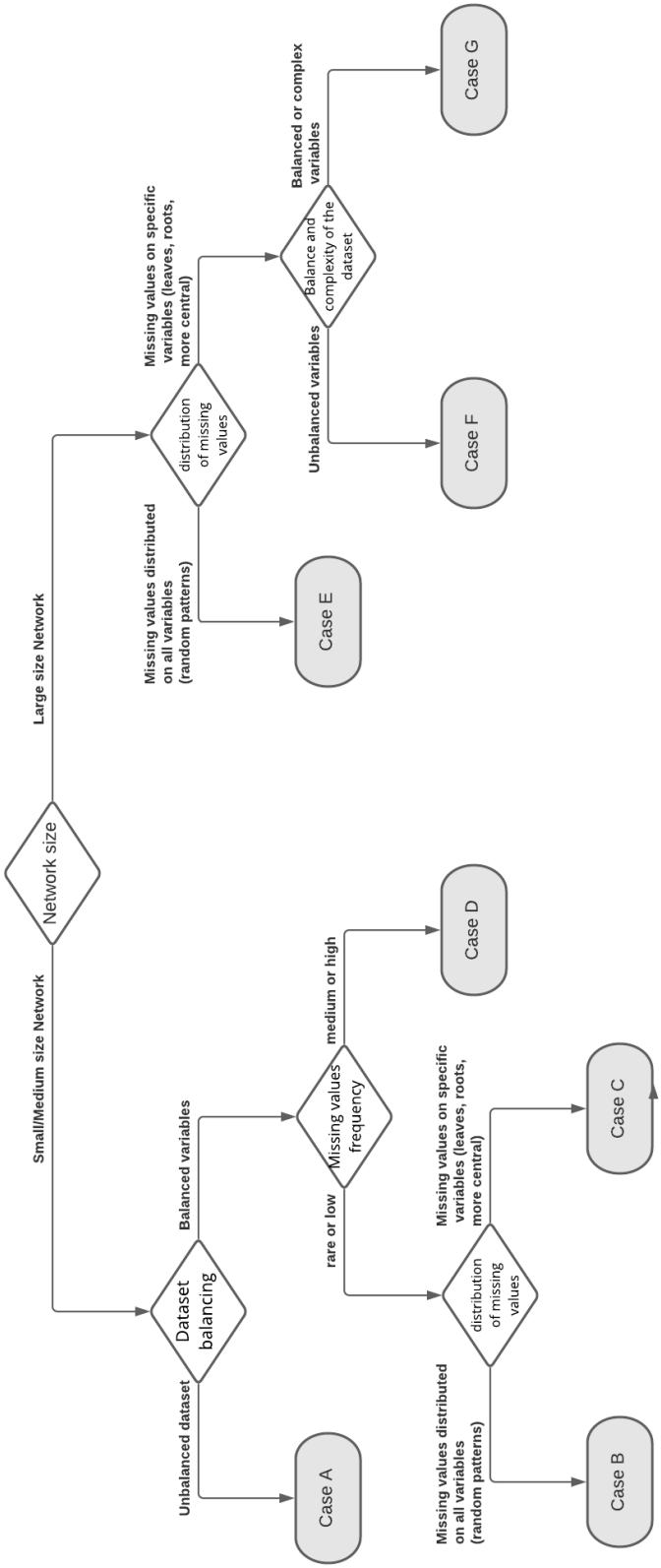
eg

eh

Appendix B

Appendix - Tree to select the best EM algorithm

This tree is relevant whenever you have a partial dataset and you want apply a specific version of the EM algorithm. We report the schema in the next page.


Network size
Small/Medium size Network
Large size Network
Dataset balancing
Balanced variables
Unbalanced dataset
Case A
Missing values frequency
medium or high
rare or low
Case D
distribution of missing values
Missing values on specific variables (leaves, roots, more central)
Missing values distributed on all variables (random patterns)
Case C
Case B
distribution of missing values
Missing values on specific variables (leaves, roots, more central)
Missing values distributed on all variables (random patterns)
Case E
Balance and complexity of the dataset
Balanced or complex variables
Unbalanced variables
Case G
Case F

Bibliography

[1] Will Badr. *6 Different Ways to Compensate for Missing Values In a Dataset*. URL: https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779.

[2] Derrick A Bennett. "How can I deal with missing data in my study?" In: *Australian and New Zealand journal of public health* 25.5 (2001), pp. 464–469.

[3] Martin Bland. *An Introduction to Medical Statistics*. URL: https://www-users.york.ac.uk/~mb55/intro/typemiss4.htm.

[4] Jason Brownlee. *How to Calculate the KL Divergence for Machine Learning*. URL: https://machinelearningmastery.com/divergence-between-probability-distributions/.

[5] Stef van Buuren. *Generate Missing Data For Simulation Purposes*. URL: https://www.rdocumentation.org/packages/mice/versions/3.10.0.1/topics/ampute.

[6] Huimin Chai, Jiangnan Lei, and Min Fang. "Estimating bayesian networks parameters using EM and Gibbs sampling". In: *Procedia computer science* 111 (2017), pp. 160–166.

[7] AC Constantinou et al. "The Bayesys data and Bayesian network repository". In: *Queen Mary University of London* (2020), pp. 2–2.

[8] Edutecnica. *Varianza*. URL: http://www.edutecnica.it/calcolo/varianza/varianza.htm.

[9] *EM Imputation and Missing Data: Is Mean Imputation Really so Terrible?* https://www.theanalysisfactor.com/em-imputation-and-missing-data-is-mean-imputation-really-so-terrible/. KAREN GRACE-MARTIN.

[10] Diane L Fairclough. *Design and analysis of quality of life studies in clinical trials*. CRC press, 2010.

[11] N. Friedman. "The Bayesian Structural EM Algorithm". In: *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. 1998, pp. 129–138.

[12] John W Graham. "Missing data analysis: Making it work in the real world". In: *Annual review of psychology* 60 (2009), pp. 549–576.

[13] Shelmith Nyagathiri Kariuki, Anthony Waititu Gichuhi, and Anthony Kibira Wanjoya. "Comparison of methods of handling missing data: a case study of KDHS 2010 data". In: *American Journal of Theoretical and Applied Statistics* 4.3 (2015), pp. 192–200.

[14] Daphne Koller. *Probabilistic Graphical Models 3: Learning*. Standorf Univerisity on Coursera. URL: https://www.coursera.org/lecture/probabilistic-graphical-models-3-learning/expectation-maximization-intro-YVVxw.

[15] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN: 0262013193.

[16] SB Kotsiantis, Dimitris Kanellopoulos, and PE Pintelas. "Data preprocessing for supervised leaning". In: *International Journal of Computer Science* 1.2 (2006), pp. 111–117.

[17] Joachim Listing and Rainer Schlittgen. "A nonparametric test for random dropouts". In: *Biometrical Journal: Journal of Mathematical Methods in Biosciences* 45.1 (2003), pp. 113–127.

[18] Joachim Listing and Rainer Schlittgen. "Tests if dropouts are missed at random". In: *Biometrical Journal: Journal of Mathematical Methods in Biosciences* 40.8 (1998), pp. 929–935.

[19] Roderick JA Little. "A test of missing completely at random for multivariate data with missing values". In: *Journal of the American statistical Association* 83.404 (1988), pp. 1198–1202.

[20] Robert Ness Marco Scutari. *Bayesian Network Structure Learning, Parameter Learning and Inference*. URL: https://www.bnlearn.com/documentation/bnlearn-manual.pdf.

[21] Robert Ness Marco Scutari. *Understanding Bayesian Networks with Examples in R*. Department of Statistics University of Oxford, 2017, pp. 139–143. URL: https://www.bnlearn.com/about/teaching/slides-bnshort.pdf.

[22] Milind Paradkar. *R Best Practices: R you writing the R way!* URL: https://www.r-bloggers.com/r-best-practices-r-you-writing-the-r-way/.

[23] Baijayanta Roy. *All About Missing Data Handling*. URL: https://towardsdatascience.com/all-about-missing-data-handling-b94b8b5d2184.

[24] Marco Scutari. *Bayesian Network Repository*. URL: https://www.bnlearn.com/bnrepository/.

[25] Marco Scutari and Robert Ness. "bnlearn: Bayesian network structure learning, parameter learning and inference". In: *R package version* 3 (2012).

[26] Peter M Fayers Shona Fielding and Craig R Ramsay. *Investigating the missing data mechanism in quality of life outcomes: a comparison of approaches*. URL: https://hqlo.biomedcentral.com/articles/10.1186/1477-7525-7-57#article-info.

[27] Chong Wu et al. *On the Convergence of the EM Algorithm: A Data-Adaptive Analysis*. 2016. arXiv: 1611.00519 [math.ST].