Structural-EM for Learning PDG Models from Incomplete Data

Jens D. Nielsen[*,a], Rafael Rumí[b], Antonio Salmerón[b]

[a]*Department of Computer Science, University of Castilla-La Mancha, Campus Universitario Parque Científico y Tecnológico s/n, 02071 Albacete (Spain)*
[b]*Department of Statistics and Applied Mathematics, University of Almería, La Cañada de San Urbano s/n, 04120 Almería (Spain)*

Abstract

Probabilistic Decision Graphs (PDGs) are a class of graphical models that can naturally encode some context specific independencies that cannot always be efficiently captured by other popular models, such as Bayesian Networks. Furthermore, inference can be carried out efficiently over a PDG, in time linear in the size of the model. The problem of learning PDGs from data has been studied in the literature, but only for the case of complete data. We propose an algorithm for learning PDGs in the presence of missing data. The proposed method is based on the Expectation-Maximisation principle for estimating the structure of the model as well as the parameters. We test our proposal on both artificially generated data with different rates of missing cells and real incomplete data. We also compare the PDG models learnt by our approach to the commonly used Bayesian Network (BN) model. The results indicate that the PDG model is less sensitive to the rate of missing data than BN model. Also, though the BN models usually attain higher likelihood, the PDGs are close to them also in size, which makes the learnt PDGs preferable for probabilistic inference purposes.

Key words: Machine Learning, Graphical Models, Learning from Incomplete Data

1. Introduction

The Probabilistic Decision Graph (PDG) model was first introduced by Bozga and Maler [1], and was originally proposed as an efficient representation of probabilistic transition systems. In this study, we consider the more general version of PDGs proposed by Jaeger [8].

[*]Corresponding author. Address: Instituto de Investigación en Informática de Albacete - I^3A, Campus Universitario, Parque Científico y Tecnológico s/n. 02071 Albacete, Spain.
Tlf.: (+34) 967 599 200 Ext. 2677, Fax: (+34) 967 599 343

Email addresses: `dalgaard@dsi.uclm.es` (Jens D. Nielsen), `rrumi@ual.es` (Rafael Rumí), `antonio.salmeron@ual.es` (Antonio Salmerón)

PDGs constitute a class of probabilistic graphical models that can represent some context specific independencies that can not efficiently be captured by conventional directed or undirected graphical models, usually called Markov Network and Bayesian Network (BN) models respectively. Furthermore, probabilistic inference can be carried out directly in the PDG structure and has a time complexity linear in the size of the PDG model. This makes learning of PDGs especially interesting, as we are learning directly the inference structure, which is in contrast to the usual scenario when learning general graphical models.

The performance of the PDG model w.r.t. general probability estimation has previously been studied and results suggest that the model in general performs competitively when compared to BN or Naïve BN models [14]. The PDG model has also been successfully applied to supervised classification problems [15, 16].

In this paper we are concerned with learning PDG models from data. This problem has been addressed by Jaeger et al. [9], where an algorithm based on the optimisation of a score is proposed for learning from complete data. However, the task of learning PDG models in the presence of incomplete data has not yet been explored in the literature. The difficulty arises in the computation of the score for a model given the database with missing values. A similar problem is found in the case of learning BN models from incomplete databases. Friedman [6] addressed this problem by proposing an algorithm for estimating the structure of a BN model based on the Expectation-Maximisation (EM) principle [5, 10].

We propose an algorithm for learning PDG models that builds on the algorithm of Friedman [6] based on the EM principle. Both the structure and the parameters are re-adjusted in each iteration of the algorithm. That is, the adjustments made to the structure are guided by the expected increase in some score metric, while the adjustments made to the parameters are guided by the expected likelihood of a completed version of the incomplete data.

2. Background and Notation

We will denote random variables by uppercase letters, e.g. X, and sets with boldface uppercase letters, e.g. $\mathbf{X}$. When X_i is a discrete categorical random variable, we will by lowercase letter $x_{i,j}$ refer to the j'th state of X_i under some ordering. We will by $R(X_i)$ refer to the set of possible states of X_i, and by $R(\mathbf{X}) = \times_{X_i \in \mathbf{X}} R(X_i)$ when $\mathbf{X}$ is a set of discrete categorical variables. We will use r_i as a shorthand for $|R(X_i)|$. By lowercase bold letters we refer to joint states of sets of variables, e.g. $\mathbf{x} \in R(\mathbf{X})$. When $X_i \in \mathbf{X}$ and $\mathbf{x} \in R(\mathbf{X})$ we denote $\mathbf{x}[X_i]$ the projection of $\mathbf{x}$ onto coordinate X_i.

Let $G = \langle \mathbf{V}, \mathbf{E} \rangle$ be a directed graph structure with set of vertices $\mathbf{V} = \{V_1, \dots, V_n\}$ and set of directed edges $\mathbf{E} \subset \mathbf{V} \times \mathbf{V}$. We will then by $ch_G(V_i)$ and $pa_G(V_i)$ refer the set of children of V_i and parents of V_i respectively in structure G, hence $ch_G(V_i) = \{V_j \in \mathbf{V} : (V_i, V_j) \in \mathbf{E}\}$ and $pa_G(V_i) = \{V_j \in \mathbf{V} : (V_j, V_i) \in \mathbf{E}\}$. A directed graph strucuture is a directed *acyclic* graph (DAG) structure if it contains no directed cycles. A *rooted* DAG is a DAG where a unique vertex V_r is without parents ($pa_G(V_r) = \emptyset$) while all other vertices have at least one parent. A tree is a rooted DAG where all vertices except the root has exactly one parent. A forest structure is a set of such trees.

2.1. The Probabilistic Decision Graph Model

A PDG encodes a joint probability distribution over a set of categorical random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ by a factorisation defined by a structure over a set of local distributions.

Definition 2.1 (The PDG Structure). *Let F be a forest structure over discrete categorical random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. A* PDG-structure *$G = \langle \mathbf{V}, \mathbf{E} \rangle$ for $\mathbf{X}$ w.r.t. F is a set of* rooted *acyclic directed graphs over nodes $\mathbf{V}$, such that:*

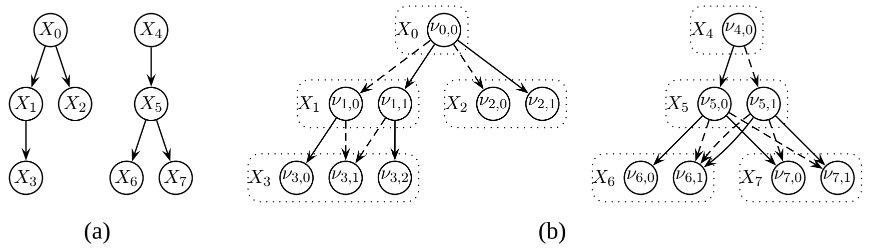
1. *Each node $\nu \in \mathbf{V}$ represents a unique $X_i \in \mathbf{X}$ and all $X_i \in \mathbf{X}$ are represented by at least one node $\nu \in \mathbf{V}$. We will by $\nu_{i,j}$ refer to the j'th node representing X_i under some ordering of the set of nodes representing X_i.*
2. *For each node $\nu_{i,j}$, each possible state $x_{i,h}$ of X_i and each successor $X_k \in ch_F(X_i)$ there exists exactly* one *edge $(\nu_{i,j}, \nu_{k,l}) \in \mathbf{E}$ with label $x_{i,h}$, where $\nu_{k,l}$ is some node representing X_k.*

Let $X_k \in ch_F(X_i)$. By $succ(\nu_{i,j}, X_k, x_{i,h})$ we refer to the unique node $\nu_{k,l}$ representing X_k that is reached from $\nu_{i,j}$ by following the edge with label $x_{i,h}$.

Example 2.1. *A forest F over binary variables $\mathbf{X} = \{X_0, \ldots, X_7\}$ can be seen in Figure 1(a), and a PDG structure over $\mathbf{X}$ w.r.t. F in Figure 1(b). The labelling of nodes in the PDG-structure is indicated in subscripts and (redundant) by the dashed boxes, e.g., the nodes representing X_2 are $\{\nu_{2,0}, \nu_{2,1}\}$. Dashed edges correspond to edges labelled 0 and solid edges correspond to edges labelled 1, for instance $succ(\nu_{5,0}, X_6, 0) = \nu_{6,1}$.*

A PDG structure is instantiated by assigning to every node a local probability distribution over the variable that it represents. By a PDG model over discrete random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ we refer to a pair $\mathcal{G} = \langle G, \mathbf{\Theta} \rangle$ where G is a PDG structure over $\mathbf{X}$ and $\mathbf{\Theta}$ is an instantiation of G. We denote by $\mathbf{p}^{\nu_{i,j}}$ the local distribution assigned to node $\nu_{i,j}$, and by $p^{\nu_{i,j}}_{x_{i,h}}$ the probability for state $x_{i,h}$ in local distribution $\mathbf{p}^{\nu_{i,j}}$. The semantics of the local distribution $\mathbf{p}^{\nu_{i,j}}$ is defined by the path(s) leading to the node $\nu_{i,j}$ from the root, that is, how $\nu_{i,j}$ can be *reached*. Let G be a PDG structure over variables $\mathbf{X}$ w.r.t. forest F. A node $\nu_{i,j}$ in G is *reached* by $\mathbf{x} \in R(\mathbf{X})$ if

- $\nu_{i,j}$ is a root in G, or
- $X_i \in ch_F(X_k)$, $\nu_{k,l}$ is reached by $\mathbf{x}$ and $\nu_{i,j} = succ(\nu_{k,l}, X_i, \mathbf{x}[X_k])$.



$\mathbf{p}^{\nu_{0,0}} = P(X_0)$	$\mathbf{p}^{\nu_{3,0}} = P(X_3 \mid X_0 = 0, X_1 = 1)$	$\mathbf{p}^{\nu_{5,1}} = P(X_5 \mid X_4 = 0)$
$\mathbf{p}^{\nu_{1,0}} = P(X_1 \mid X_0 = 0)$	$\mathbf{p}^{\nu_{3,1}} = P(X_3 \mid X_1 = 0)$	$\mathbf{p}^{\nu_{6,0}} = P(X_6 \mid X_4 = 1, X_5 = 1)$
$\mathbf{p}^{\nu_{1,1}} = P(X_1 \mid X_0 = 1)$	$\mathbf{p}^{\nu_{3,2}} = P(X_3 \mid X_0 = 1, X_1 = 1)$	$\mathbf{p}^{\nu_{6,1}} = P(X_6 \mid X_4 = 0 \vee \{X_4 = 1, X_5 = 0\})$
$\mathbf{p}^{\nu_{2,0}} = P(X_2 \mid X_0 = 0)$	$\mathbf{p}^{\nu_{4,0}} = P(X_4)$	$\mathbf{p}^{\nu_{7,0}} = P(X_7 \mid X_4 = X_5)$
$\mathbf{p}^{\nu_{2,1}} = P(X_2 \mid X_0 = 1)$	$\mathbf{p}^{\nu_{5,0}} = P(X_5 \mid X_4 = 1)$	$\mathbf{p}^{\nu_{7,0}} = P(X_7 \mid X_4 \neq X_5)$

(c)

Figure 1: A forest F over binary variables $\mathbf{X} = \{X_0, \ldots, X_7\}$ is shown in (a), and a PDG-structure over $\mathbf{X}$ w.r.t. variable forest F is shown in (b). In the PDG-structure in (b), solid edges are labelled with value 1 and dashed edges are labelled with value 0. In (c), we have indicated the probabilistic interpretation of the parameters for each node in the PDG structure of (b).

By $reach_G(i, \mathbf{x})$ we denote the unique node representing X_i reached by $\mathbf{x}$ in PDG-structure G.

A PDG model $\mathcal{G} = \langle G, \mathbf{\Theta} \rangle$ over variables $\mathbf{X}$ represents a joint distribution $P^{\mathcal{G}}$ by the following factorisation:

$$P^{\mathcal{G}}(\mathbf{x}) = \prod_{X_i \in \mathbf{X}} p_{\mathbf{x}[X_i]}^{reach_G(i,\mathbf{x})}. \tag{1}$$

Example 2.2. *To instantiate the PDG structure in Fig. 1(b), we assign a local distribution to each node in the structure with the probabilistic interpretation given in Fig. 1(c). We can read some context specific independencies off this table, e.g. X_6 is independent of X_5 only in the context $X_4 = 0$.*

2.2. Selecting PDG models using complete data

For assessing models in the presence of observed data, we can use a penalised likelihood score function. Let $\mathcal{G}$ be a PDG model over variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ and let $\mathbf{D}$ be a set of N complete observations of $\mathbf{X}$, then we define a general score function as:

$$S_\lambda(\mathbf{D}, \mathcal{G}) = (1 - \lambda) \cdot L(\mathbf{D}, \mathcal{G}) - \lambda \cdot size(\mathcal{G}), \tag{2}$$

where $0 < \lambda < 1$, $size(\mathcal{G})$ is some measure of complexity of $\mathcal{G}$ and $L(\mathbf{D}, \mathcal{G})$ is the log-likelihood of $\mathbf{D}$ given $\mathcal{G}$. A typical definition of $size(G)$ is the number of free parameters in model $\mathcal{G}$. Assume that data $\mathbf{D}$ is a set of i.i.d samples of some (unknown) multivariate distribution over the n categorical random variables $\mathbf{X} = \{X_1, \ldots X_n\}$. We can use the following notation, $\mathbf{D} = \{x_i^k : 1 \leq i \leq n, 1 \leq k \leq N\}$, where x_i^k is the sampled value of variable X_i in row k. By $\mathbf{x}^k = \{x_1^k, \ldots, x_n^k\}$ we will refer to the kth row of $\mathbf{D}$. Then the log-likelihood

$L(\mathbf{D}, \mathcal{G})$ is:

$$\begin{aligned} L(\mathbf{D}, \mathcal{G}) &= \log \prod_{k=1}^{N} P^{\mathcal{G}}(\mathbf{X} = \mathbf{x}^k) = \sum_{k=1}^{N} \log P^{\mathcal{G}}(\mathbf{X} = \mathbf{x}^k) && (3) \\ &= \sum_{i=1}^{n} \sum_{h=1}^{r_i} \sum_{j=1}^{v_i} \#_{\mathbf{D}}(x_{i,h}, \nu_{i,j}) \log p_{x_{i,h}}^{\nu_{i,j}} \,, && (4) \end{aligned}$$

where v_i is the number of nodes representing X_i and $\#_{\mathbf{D}}(E)$ is the count of instances in $\mathbf{D}$ satisfying requirement E. For example, in Eq. (4) $\#_{\mathbf{D}}(x_{i,h}, \nu_{i,j})$ is the count of data items in $\mathbf{D}$ where variable X_i is observed in state $x_{i,h}$ and where the $\nu_{i,j}$ is reached.

3. Learning from Incomplete Data

When data is incomplete and the values for some variables are missing in some of the rows in our database, it becomes problematic to compute the likelihood in Eq. (3) as not all n variables are always observed. We will augment the database with a special "?"-state whenever a variable is not observed, and will then use the notation $\mathbf{D} = \mathbf{D}_O \cup \mathbf{D}_M$, where $\mathbf{D}_O = \{x_i^k \in \mathbf{D} : x_i^k \neq ?\}$ is the set of elements of $\mathbf{D}$ containing a value and $\mathbf{D}_M = \{x_i^k \in \mathbf{D} : x_i^k = ?\} = \mathbf{D} \setminus \mathbf{D}_O$. Furthermore, let $\mathbf{X}_O^k = \{X_i : x_i^k \in \mathbf{D}_O\}$ be the set of variables observed in the kth row, let $\mathbf{X}_M^k = \mathbf{X} \setminus \mathbf{X}_O^k$ and let $\mathbf{x}_O^k = \{x_i^k \in \mathbf{D}_O\}$ be the observations of $\mathbf{X}_O^k$. Then, Eq. (3) becomes:

$$\begin{aligned} L(\mathbf{D}, \mathcal{G}) &= \log \prod_{k=1}^{N} P^{\mathcal{G}}(\mathbf{X}_O^k = \mathbf{x}_O^k) && (5) \\ &= \log \prod_{k=1}^{N} \sum_{\mathbf{x}' \in R(\mathbf{X}_M^k)} P^{\mathcal{G}}(\mathbf{X}_O^k = \mathbf{x}_O^k, \mathbf{X}_M^k = \mathbf{x}') \,. && (6) \end{aligned}$$

The most typical approach to learn from incomplete data is to apply the Expectation Maximisation (EM) principle (see [12, 5]). In the following Sec. 3.1 we review this approach to structural learning in general. Before doing this, however, we will introduce a few simpler approaches that will later be used for comparison when evaluating our proposal experimentally in Sec. 6.

A simple approach to avoiding the exponentially large sum of Eq. (6) is to use an available-case-analysis (ACA) approximation to the complete case formula of Eq. (4). In ACA one uses as much of the available observed data as possible. So, considering the structure of Fig. 1, any row in the database that contains the observations $X_1 = 0$ and $X_3 = 0$ would increment the count $\#_{\mathbf{D}}(x_{3,0}, \nu_{3,1})$ regardless of the missing values, as any instance with $X_1 = 0$ reaches $\nu_{3,1}$.

An even simpler approach to learning from incomplete data is complete-case-analysis (CCA) where any row in the database that misses values for some variables is removed from the database and any method that requires complete data can then subsequently be applied.

Algorithm 1 The structural EM procedure

```
 1: procedure SEM($\mathbf{D}$)
 2:     Let $\mathcal{G}^0 = \langle G^0, \mathbf{\Theta}^0 \rangle$ be the initial model.
 3:     $n \leftarrow 0$
 4:     repeat
 5:         $\mathbf{\Theta}^{n+1} \leftarrow \underset{Legal\ \mathbf{\Theta}}{\textit{argmax}}\ Q(\langle G^n, \mathbf{\Theta} \rangle, \mathbf{D}|\mathbf{D}_O, \mathcal{G}^n)$
 6:         $G^{n+1} \leftarrow \underset{G \in \mathcal{N}(G^n)}{\textit{argmax}}\ Q(\langle G, \cdot \rangle, \mathbf{D}|\mathbf{D}_O, \langle G^n, \mathbf{\Theta}^{n+1} \rangle)$
 7:         $\mathcal{G}^{n+1} \leftarrow \langle G^{n+1}, \mathbf{\Theta}^{n+1} \rangle$
 8:         $n \leftarrow n + 1$
 9:     until $Q(\mathcal{G}^n, \mathbf{D}|\mathbf{D}_O, \mathcal{G}^{n-1}) \leq Q(\mathcal{G}^{n-1}, \mathbf{D}|\mathbf{D}_O, \mathcal{G}^{n-1})$
10:     return $\mathcal{G}^{n-1}$
```

3.1. The EM principle in structural learning

If we view the missing part $\mathbf{D}_M$ of incomplete data $\mathbf{D}$ as a random variable governed by (unknown) distribution P^*, we can compute the expected log likelihood of $\mathbf{D}$ in model $\mathcal{G}$ as:

$$E[L(\mathbf{D}, \mathcal{G})|\mathbf{D}_O, P^*] = \sum_{i=1}^{n} \sum_{h=1}^{r_i} \sum_{j=1}^{v_i} E[\#_{\mathbf{D}}(x_{i,h}, \nu_{i,j})] \log p_{x_{i,h}}^{\nu_{i,j}} , \tag{7}$$

where the second expectation also is with respect to $\mathbf{D}_O$ and P^*. In a structural EM algorithm like the one proposed by Friedman [6], we optimise the expected likelihood instead of directly optimising the likelihood which in the presence of incomplete data no longer decomposes. Decomposability of the likelihood is important for model selection in which the search procedure in each step evaluates candidate models from a neighbourhood that is generated from a current model by local transformation. We define the expected score as a function Q:

$$Q(\mathcal{G}, \mathbf{D}|\mathbf{D}_O, \mathcal{G}^*) = (1 - \lambda)E[L(\mathbf{D}, \mathcal{G})|\mathbf{D}_O, \mathcal{G}^*] - \lambda size(\mathcal{G}) \ . \tag{8}$$

In Eq. (8) we use the current model $\mathcal{G}^*$ as the reference distribution P^*. The structural EM procedure can now be stated as in Algorithm 1.

First, in line 5 of Alg. 1 we basically need to find MAP parameters for $\mathcal{G}$. Exact methods are usually intractable, so normally some approximation method is employed. Originally, Friedman [6] proposed to use a standard EM approach in this step while Peña et al. [17] propose as a more computationally efficient alternative to use the *branch and bound* procedure of Ramoni and Sebastiani [18]. However, the choice of approach in this step is not crucial to the following discussion of Sec. 4.

Second, in line 6 of Alg. 1, the function $\mathcal{N}(\cdot)$ is the neighbourhood generating function. We will define simple split and merge operations that implement structural modifications for generating neighbours from a current PDG model $\mathcal{G}$. We will explain how to compute the expectations needed to evaluate the expected score of a neighbour.

4. Structual EM for PDG Models

In this section we will explain how Alg. 1 can be applied to PDG models. First, for constructing an initial model we need a forest structure over the variables in the domain. We accomplish this using the classical algorithm of Chow and Liu [4]. This algorithm induces a maximum weight spanning tree using mutual information as the edge-weights. In Sec. 5 we explain how to compute mutual information from incomplete data.

Inducing the initial tree by finding a maximum weight spanning tree using mutual information as edge-weights is different from previously proposed approaches for inducing variable forests/trees. In [9] a χ^2 test for conditional independence is used to assign marginally independent variables in different trees and conditionally independent variables in different sub-trees. In this study we use the above mentioned mutual information based method as it is less data-intensive compared to repeated χ^2 tests. Also, the computation of the test statistic for the χ^2 test is problematic when the data is incomplete, and it is known that the type II error associated with the test is high if the data is scarce, which may increase the risk of acceptance of independencies that are not supported by the data. Restricting the forest to a single tree does not limit the expressivity of the models that can be obtained, but it may produce models with a higher number of parameters.

Assuming that we have the initial tree structure F over the variables, we initialise a PDG model as follows: for every variable $X_i \in \mathbf{X}$ with $pa_F(X_i) = X_k$, we create $v_i = r_k$ new nodes $\{\nu_{i,1}, \nu_{i,2}, \ldots, \nu_{i,v_i}\}$ representing X_i. We then connect every node $\nu_{k,j}$ representing X_k such that $succ(\nu_{k,j}, X_i, x_{k,z}) = \nu_{i,z}$. That is, for state $x_{k,z}$ of variable X_k the node $\nu_{i,z}$ is always reached. Constructing the initial PDG model in this way allows every variable to be modelled as marginally dependent on its parent and its set of children in F.

Finally, we use a random parametrisation $\mathbf{\Theta}^0$ of the initial structure G^0.

4.1. The Neighbourhood of a Model

In this subsection we explain how the neighbourhood $\mathcal{N}(G)$ of a PDG structure G is generated. We include operations that work on the PDG structure only, and leave the structure over the variables fixed. Operations that change the structure over the variables (e.g. operations that swap the position of two variables) are problematic as they potentially require the creation of a lot of new node connections. Offhand, it is not intuitive to us how to best do this in general, and therefore we choose to focus on the following two less dramatic structural changes, namely splitting and merging parameter nodes. These operations have both previously been used by Jaeger et al. [9] for learning in the case of complete data. Jaeger et al. [9] use an additional third operator that redirects edges. We leave this operator out of the algorithm for simplicity.

Merging Nodes. The merge operator takes a pair of nodes $\{\nu_{i,a}, \nu_{i,b}\}$ representing the same variable X_i. The nodes $\nu_{i,a}$ and $\nu_{i,b}$ are selected such that for any state

$x_{i,h} \in R(X_i)$ and child $X_j \in ch_F(X_i)$, $succ(\nu_{i,a}, X_j, x_{i,h}) = succ(\nu_{i,b}, X_j, x_{i,h})$. The merge operation then simply consists in replacing nodes $\nu_{i,a}$ and $\nu_{i,b}$ with a new node $\nu_{i,c}$, where $\nu_{i,c}$ has as children exactly the children of $\nu_{i,a}$ (or $\nu_{i,b}$) and as parents inherits the union of parents of $\nu_{i,a}$ and parents of $\nu_{i,b}$.

Splitting Nodes. The splitting operator takes as input a single node $\nu_{i,j}$ with m parents where $2 \leq m$, and replaces $\nu_{i,j}$ with m new nodes all representing X_i. Each new node inherits all the children of $\nu_{i,j}$, and exactly one unique parent of $\nu_{i,j}$.

4.2. The candidate-space

Before continuing with the further development of the algorithm, we will investigate the properties of the space of candidate models that we can reach with the split and merge operations as defined in Sec. 4.1.

We formally define the neighbourhood $\mathcal{N}$ as follows:

Definition 4.1. *Let $\mathscr{F}$ be a set of functions $f : \mathbf{G} \rightarrow \mathbf{G}$ where $\mathbf{G}$ is the set of all valid PDG structures according to syntactical definition 2.1. Then the neighbourhood $\mathcal{N}_{\mathscr{F}}$ of $G \in \mathbf{G}$ is defined as:*

$$\mathcal{N}_{\mathscr{F}}(G) = \{G' : \exists f \in \mathscr{F}[f(G) = G']\} . \tag{9}$$

The candidate-space $\mathcal{C}$ is defined as the set of models that can be reached by traversing neighbourhoods.

Definition 4.2. *Let $\mathcal{F}$ and $\mathbf{G}$ be as in Def. 4.1, then the candidate space $\mathcal{C}_{\mathscr{F}}$ given an initial model $G_0 \in \mathbf{G}$ is defined as:*

$$\mathcal{C}_{\mathscr{F}}(G_0) = \{G' : G' \in \mathcal{N}_{\mathscr{F}}(G_0) \vee \exists G^*[G^* \in \mathcal{N}_{\mathscr{F}}(G_0) \wedge G' \in \mathcal{C}_{\mathscr{F}}(G^*)]\} . \tag{10}$$

Lemma 4.1. *Let $\mathbf{G}_F$ the set of PDG structures with the same underlying variable-forest F, let $\mathscr{F} = \{\mathrm{merge}, \mathrm{split}\}$ where merge and split are the operations defined in Sec. 4.1. Let G_0 be an arbitrary PDG structure with variable-forest F, then:*

$$\mathcal{C}_{\mathscr{F}}(G_0) = \mathbf{G}_F . \tag{11}$$

Proof: We will show that for any two PDG structures G and G^* that respects the same underlying variable forest F, we can transform G into G^* by a series of merge and split operations.

The first step in the transformation is to merge all nodes in G from the leaves and up to the root. This results in a structure in which every variable is represented by exactly one unique node. Then, from the root and down to the leafs we do the following for each variable X:

1. split the single node representing X in G
2. merge the nodes representing X in G to reconstruct the local structure between X and its parent in G^*

$\square$

Lemma 4.1 says that any valid PDG structure within the same variable structure is reachable from the initial model.

4.3. Scoring a Neighbour Model

In this section we detail how to compute the score $Q(\langle G', \cdot\rangle, \mathbf{D}|\mathbf{D}_O, \langle G, \mathbf{\Theta}\rangle)$ of a neighbour $G' \in \mathcal{N}(G)$ generated by merging two nodes or splitting a node. In fact, we will not compute the full expected score, but only the terms that are different.

4.3.1. Scoring a Merge Operation

Assume PDG $\mathcal{G}'$ is constructed from PDG $\mathcal{G} = \langle G, \Theta\rangle$ by merging nodes $\nu_{i,a}, \nu_{i,b} \in \mathbf{V}_i$ in structure G. Let the node $\nu_{i,c}$ be the one replacing $\nu_{i,a}$ and $\nu_{i,b}$ in $\mathcal{G}'$, and assume that we have computed (and stored) all expected counts for all nodes. Then, computing the expected counts for model $\mathcal{G}'$ under distribution $P^{\mathcal{G}}$ reduces to computing expected counts for $\nu_{i,c}$, which can be done efficiently from expectations $\#_{\mathbf{D}}(x_{i,h}, \nu_{i,a})$ and $\#_{\mathbf{D}}(x_{i,h}, \nu_{i,b})$, that is :

$$E[\#_{\mathbf{D}}(x_{i,h}, \nu_{i,c})|\mathbf{D}_O, \mathcal{G}] = \\ E[\#_{\mathbf{D}}(x_{i,h}, \nu_{i,a})|\mathbf{D}_O, \mathcal{G}] + E[\#_{\mathbf{D}}(x_{i,h}, \nu_{i,b})|\mathbf{D}_O, \mathcal{G}]\,. \quad (12)$$

Hence, computing the difference in expected score $\Delta Q_{merge}(\nu_{i,a}, \nu_{i,b})$ reduces to computing the difference between the terms of the expected score involving nodes $\nu_{i,a}$ and $\nu_{i,b}$ and the new node $\nu_{i,c}$:

$$\Delta Q_{merge}(\nu_{i,a}, \nu_{i,b}) = Q(\mathcal{G}', \mathbf{D}|\mathbf{D}_O, \mathcal{G}) - Q(\mathcal{G}, \mathbf{D}|\mathbf{D}_O, \mathcal{G}) \\ = (1-\lambda)\left(\sum_{h=1}^{r_i} E[L_h^{\nu_{i,c}} - L_h^{\nu_{i,a}} - L_h^{\nu_{i,b}}|\mathbf{D}_O, \mathcal{G}]\right) + \lambda \cdot (r_i - 1) \quad (13)$$

where $\nu_{i,c}$ is the node resulting from merging $\nu_{i,a}$ and $\nu_{i,b}$, and $L_h^{\nu_{i,j}}$ is the term in the log-likelihood corresponding to node $\nu_{i,j}$ and the hth state of X_i. The expectation in (13) obviously can be computed term by term, and we see that $E[L_h^{\nu_{i,c}}|\mathbf{D}_O, \mathcal{G}] = E[\#_{\mathbf{D}}(x_{i,h}, \nu_{i,c})|\mathbf{D}_O, \mathcal{G}] \log E[p_{x_{i,h}}^{\nu_{i,c}}|\mathbf{D}_O, \mathcal{G}]$, where the expectation $E[p_{x_{i,h}}^{\nu_{i,c}}|\mathbf{D}_O, \mathcal{G}]$ is computed as the fraction $\frac{E[\#_{\mathbf{D}}(x_{i,h},\nu_{i,c})|\mathbf{D}_O,\mathcal{G}]}{E[\#_{\mathbf{D}}(\nu_{i,c})|\mathbf{D}_O,\mathcal{G}]}$. The count $\#_{\mathbf{D}}(\nu_{i,a})$ is just
$\sum_{h=1}^{r_i} \#_{\mathbf{D}}(x_{i,h}, \nu_{i,a})$.

The expectations $E[\#_{\mathbf{D}}(x_{i,h}, \nu_{i,j})|\mathbf{D}_O, \mathcal{G}]$ for any state $x_{i,h} \in R(X_i)$ and node $\nu_{i,j} \in \mathbf{V}_i$ are exactly the expectations we would compute in the parametric EM step in line 5 of Alg. 1. Therefore, these counts have already been computed for structure $\mathcal{G}^n$ in line 5 of Alg. 1, and can easily be made available at no extra cost.

4.3.2. *Scoring a Split Operation*

Let $\mathit{inc}(\nu_{i,j})$ be the set of edges incoming to $\nu_{i,j}$ in PDG structure $G = \langle \mathbf{V}, \mathbf{E} \rangle$, that is $\mathit{inc}(\nu_{i,j}) = \{(\nu_{k,z}, \nu_{i,j}) \in \mathbf{E}\}$. By $l^u_{\nu_{i,j}}$ we will denote the u'th element of $\mathit{inc}(\nu_{i,j})$ under some ordering. With $\nu^u_{i,j}$ we denote the node replacing $\nu_{i,j}$ for its uth incoming edge. Node $\nu_{i,j}$ is representing variable X_i and let the parent of X_i in the variable forest be X_k, hence by the definition of PDG structure, all parent nodes of $\nu_{i,j}$ represent variable X_k. The expected counts $E[\#_{\mathbf{D}}(\nu^u_{i,j}, x_{i,h})|\mathbf{D}_O, \mathcal{G}]$ for the node $\nu^u_{i,j}$ where $l^u_{\nu_{i,j}} = (\nu_{k,z}, \nu_{i,j})$ is labelled with state $x_{k,g}$ is then:

$$E[\#_{\mathbf{D}}(\nu^u_{i,j}, x_{i,h})|\mathbf{D}_O, \mathcal{G}] = E[\#_{\mathbf{D}}(\nu_{k,z}, x_{k,g}, x_{i,h})|\mathbf{D}_O, \mathcal{G}]\,. \tag{14}$$

The expectation in Eq. (14) can not be reconstructed from expected counts already computed for $\mathcal{G}$ in the structural parametric EM step of Alg. 1 (line 5) as was the case for the counts needed to evaluate a merge operation. However, anticipating that we will need such counts, we can store them during the computation of expectations in line 5 of Alg. 1. Assume that we have these expected counts available for structure G under the distribution defined by the PDG model $\mathcal{G} = \langle G, \Theta \rangle$. We can then compute the difference $\Delta Q_{\mathit{split}}(\nu_{i,j}) = Q(\mathcal{G}', \mathbf{D}|\mathbf{D}_O, \mathcal{G}) - Q(\mathcal{G}, \mathbf{D}|\mathbf{D}_O, \mathcal{G})$ for PDG model $\mathcal{G}'$ with structure G' generated by splitting node $\nu_{i,j}$ in structure G, as follows:

$$\begin{aligned}\Delta Q_{\mathit{split}}(\nu_{i,j}) &= Q(\mathcal{G}', \mathbf{D}|\mathbf{D}_O, \mathcal{G}) - Q(\mathcal{G}, \mathbf{D}|\mathbf{D}_O, \mathcal{G}) \\ &= (1-\lambda)\left[\sum_{h=1}^{r_i}\left(\sum_{u=1}^{m} E[L_h^{\nu^u_{i,j}}|\mathbf{D}_O, \mathcal{G}]\right) - E[L_h^{\nu_{i,j}}|\mathbf{D}_O, \mathcal{G}]\right] \\ &\qquad - \lambda(|\mathit{inc}(\nu_{i,j})| - 1)(r_i - 1)\,, \end{aligned} \tag{15}$$

where the log-likelihood terms $L^{\cdot}_{\cdot}$ are as described in Sec. 4.3.1. Further, it is clear that we can not split a root node as it has no parents.

4.4. *Computing the Expectations*

In order to compute the expected counts in sections 4.3.1 and 4.3.2, it is necessary to calculate probabilities of the form $P^{\mathcal{G}}(\{\nu \text{ is reached } \wedge X_i = x_i\}|\mathbf{Y} = \mathbf{y})$ for all $X_i \in \mathbf{X}$ and $\nu \in \mathbf{V}_i$, where $\mathcal{G}$ is a PDG over variables $\mathbf{X}$ and $\mathbf{y}$ is a joint observation of variables $\mathbf{Y} \subset \mathbf{X}$.

The computation of such probabilities can be done efficiently by the procedure described by Jaeger [8], which carries out the inference in time linear in the size of the PDG model. We will briefly describe this procedure in the following, and we refer the reader to Jaeger [8], Section 4 for details on PDG inference.

Broadly speaking, belief updating in a PDG $\mathcal{G}$ in the presence of evidence $\mathbf{Y} = \mathbf{y}$ is done by first restricting $\mathcal{G}$ to $\mathbf{Y} = \mathbf{y}$. A PDG $\mathcal{G}$ over variables $\mathbf{X}$ is restricted to $\mathbf{Y} = \mathbf{y}$ by setting for each variable $X_i \in \mathbf{Y}$ and each node $\nu_{i,j}$ representing $X_i \in \mathbf{Y}$ the parameter $\mathbf{p}^{\nu_{i,j}}$ to 0 for every state $x_{i,k} \neq \mathbf{y}[X_i]$ and leaving the parameter unchanged for state $\mathbf{y}[X_i]$. What we need to do to compute the probability of some subset $\mathbf{W} \subset R(\mathbf{X})$ obviously is:

$$P^{\mathcal{G}}(\mathbf{W}) = \sum_{\mathbf{w} \in \mathbf{W}} \prod_{X_i \in \mathbf{X}} p_{\mathbf{w}[X_i]}^{reach_G(i,\mathbf{w})} . \tag{16}$$

When $\mathcal{G}$ has been restricted to evidence $\mathbf{Y} = \mathbf{y}$, the probability of (16) is in fact $P^{\mathcal{G}}(\mathbf{W}, \mathbf{Y} = \mathbf{y})$. We will be particularly interested in computing (16) when $\mathbf{W}$ is the partition of $R(\mathbf{X})$ that reaches a specific node ν in the model. To do this we compute for each node in the structure parts of the product in (16), namely *in-flow* and *out-flow*. Let F denote the forest of variable trees in $\mathcal{G}$, then the out-flow is computed by the recursive formula:

$$\mathit{ofl}(\nu_{i,j}) = \sum_{x_{i,h} \in R(X_i)} p_{x_{i,h}}^{\nu_{i,j}} \prod_{Y \in ch_F(X_i)} \mathit{ofl}(succ(\nu_{i,j}, Y, x_{i,h})) . \tag{17}$$

When $\nu_{i,j}$ is a root, we compute in-flow as

$$\mathit{ifl}(\nu_{i,j}) = \prod_{\nu \neq \nu_{i,j}, \nu \text{ is root}} \mathit{ofl}(\nu) . \tag{18}$$

When $\nu_{i,j}$ is not a root and X_p is the parent of X_i in the variable forest, in-flow can be computed as:

$$\mathit{ifl}(\nu_{i,j}) = \sum_{x_{p,h} \in R(X_p)} \sum_{\substack{\nu_{p,k}: \\ \nu_{i,j} = succ(\nu_{p,k}, X_i, x_{p,h})}} [\mathit{ifl}(\nu_{p,k}) p_{x_{p,h}}^{\nu_{p,k}} \prod_{Y \in ch_F(X_p) \setminus X_i} \mathit{ofl}(succ(\nu_{p,k}, Y, x_{p,h}))] . \tag{19}$$

Computing *ofl* values is easily done recursively bottom-up in the structure, and in this traversal we store for each node $\nu_{i,j}$ and state $x_{i,j} \in R(X_i)$ the value:

$$\pi_{x_{i,h}}^{\nu_{i,j}} = \prod_{Y \in ch_F(X_i)} \mathit{ofl}(succ(\nu_{i,j}, Y, x_{i,h})) . \tag{20}$$

Now, if $\mathbf{W}$ is the set of configurations reaching node ν, then $P(\{\nu \text{ is reached}\}) = P(\mathbf{W}) = \mathit{ifl}(\nu) \cdot \mathit{ofl}(\nu)$. When *ifl*, *ofl* and π have been computed for every node in $\mathcal{G}_{\mathbf{Y}=\mathbf{y}}$ and states of the variables, we can compute $P^{\mathcal{G}}(\{\nu_{i,j} \text{ is reached } \wedge X_i = x_{i,h}\} | \mathbf{Y} = \mathbf{y})$ as:

$$P^{\mathcal{G}}(\{\nu_{i,j} \wedge x_{i,h}\} | \mathbf{Y} = \mathbf{y}) = \frac{1}{P^{\mathcal{G}}(\mathbf{Y} = \mathbf{y})} \mathit{ifl}(\nu_{i,j}) \cdot p_{x_{i,h}}^{\nu_{i,j}} \cdot \pi_{x_{i,h}}^{\nu_{i,j}} , \tag{21}$$

where $P^{\mathcal{G}}(\mathbf{Y} = \mathbf{y}) = \prod_{\nu \in \text{ roots}} \mathit{ofl}(\nu)$. The next example is aimed to illustrate the process of computing probabilities in PDGs.

Example 4.1. *Consider a PDG $\mathcal{G}$ for variables $X_0, X_1, X_2 and X_3$ and structure as in the left hand side of Fig. 1(b). Assume that the parameter nodes are instantiated with the following local distributions:*

$$\begin{array}{llll}\mathbf{p}^{\nu_{0,0}} = (0.2, 0.8) & \mathbf{p}^{\nu_{1,0}} = (0.7, 0.3) & \mathbf{p}^{\nu_{1,1}} = (0.4, 0.6) & \mathbf{p}^{\nu_{2,0}} = (0.1, 0.9) \\ \mathbf{p}^{\nu_{2,1}} = (0.8, 0.2) & \mathbf{p}^{\nu_{3,0}} = (0.6, 0.4) & \mathbf{p}^{\nu_{3,1}} = (0.3, 0.7) & \mathbf{p}^{\nu_{3,2}} = (0.5, 0.5)\end{array}$$

If we want to compute any probability like, for instance, $P^{\mathcal{G}}(\{\nu_{1,0} \wedge X_1 = 1\}|X_3 = 0)$*, the first step is to obtain* $\mathcal{G}_{X_3=0}$*, which is the restriction of* $\mathcal{G}$ *to* $X_3 = 0$*. It is computed by replacing* $\mathbf{p}^{\nu_{3,0}}$*,* $\mathbf{p}^{\nu_{3,1}}$ *and* $\mathbf{p}^{\nu_{3,2}}$ *by* $\mathbf{p}^{\nu_{3,0}} = (0.6, 0)$*,* $\mathbf{p}^{\nu_{3,1}} = (0.3, 0)$ *and* $\mathbf{p}^{\nu_{3,2}} = (0.5, 0)$*.*

Next we have to compute the outflows and finally the inflows. The outflows for the leaf parameter nodes are easy to compute: $\mathit{ofl}(\nu_{3,0}) = 0.6$*,* $\mathit{ofl}(\nu_{3,1}) = 0.3$*,* $\mathit{ofl}(\nu_{3,2}) = 0.5$*,* $\mathit{ofl}(\nu_{2,0}) = 1$ *and* $\mathit{ofl}(\nu_{2,1}) = 1$*. For the other nodes we have*

$$\mathit{ofl}(\nu_{1,0}) = 0.7 \times \mathit{ofl}(\nu_{3,1}) + 0.3 \times \mathit{ofl}(\nu_{3,0}) = 0.7 \times 0.3 + 0.3 \times 0.6 = 0.39 \ .$$

$$\mathit{ofl}(\nu_{1,1}) = 0.4 \times \mathit{ofl}(\nu_{3,1}) + 0.6 \times \mathit{ofl}(\nu_{3,2}) = 0.4 \times 0.3 + 0.6 \times 0.5 = 0.42 \ .$$

$$\begin{aligned}\mathit{ofl}(\nu_{0,0}) &= 0.2 \times \mathit{ofl}(\nu_{1,0}) \times \mathit{ofl}(\nu_{2,0}) + 0.8 \times \mathit{ofl}(\nu_{1,1}) \times \mathit{ofl}(\nu_{2,1}) \\ &= 0.2 \times 0.39 \times 1 + 0.8 \times 0.42 \times 1 = 0.414 \ .\end{aligned}$$

Now we calculate the inflows. As there is only one root, $\mathit{ifl}(\nu_{0,0}) = 1$*. The other inflows are:*

$$\mathit{ifl}(\nu_{1,0}) = \mathit{ifl}(\nu_{0,0}) \times 0.2 \times \mathit{ofl}(\nu_{2,0}) = 0.2 \ .$$

$$\mathit{ifl}(\nu_{1,1}) = \mathit{ifl}(\nu_{0,0}) \times 0.8 \times \mathit{ofl}(\nu_{2,1}) = 0.8 \ .$$

$$\mathit{ifl}(\nu_{2,0}) = \mathit{ifl}(\nu_{0,0}) \times 0.2 \times \mathit{ofl}(\nu_{1,0}) = 1 \times 0.2 \times 0.39 = 0.078 \ .$$

$$\mathit{ifl}(\nu_{2,1}) = \mathit{ifl}(\nu_{0,0}) \times 0.8 \times \mathit{ofl}(\nu_{1,1}) = 1 \times 0.8 \times 0.42 = 0.336 \ .$$

$$\mathit{ifl}(\nu_{3,0}) = \mathit{ifl}(\nu_{1,0}) \times 0.3 = 0.2 \times 0.3 = 0.06 \ .$$

$$\mathit{ifl}(\nu_{3,1}) = \mathit{ifl}(\nu_{1,0}) \times 0.7 + \mathit{ifl}(\nu_{1,1}) \times 0.7 = 0.2 \times 0.7 + 0.8 \times 0.7 = 0.7 \ .$$

Now, we have that the probability of the observation is the outflow stored in the root parameter node, that is, $P(X_3 = 0) = 0.414$*, and therefore we can compute the probability we were looking for as follows:*

$$\begin{aligned}P^{\mathcal{G}}(\{\nu_{1,0} \wedge X_1 = 1\}|X_3 = 0) &= \frac{1}{0.414} \times 0.2 \times 0.3 \times \mathit{ofl}(\nu_{3,0}) \\ &= \frac{0.2 \times 0.3 \times 0.6}{0.414} = 0.869 \ .\end{aligned}$$

At this point we have all the necessary to compute the expected counts as follows:

$$E[\#_{\mathbf{D}}(x_{i,h}, \nu_{i,j})|\mathbf{D}_O, \mathcal{G}] = \sum_{d=1}^{N} P^{\mathcal{G}}(\{\nu_{i,j} \wedge x_{i,h}\}|\mathbf{Y}_d = \mathbf{y}_d) \,, \tag{22}$$

where $\mathbf{Y}_d$ is the set of observed variables in the d-th data case, and $\mathbf{y}_d$ is the value observed for $\mathbf{Y}_d$ in the d-th data case. Notice that $P^{\mathcal{G}}$ is equal to 1 if, in record d, X_i is observed to its value $x_{i,h}$ and $\nu_{i,j}$ is reached, and is equal to 0 if it is observed to a different value or $\nu_{i,j}$ is not reached. If the value for X_i is missing in record d, then $P^{\mathcal{G}}$ is computed as in Eq. (21).

The probability $P^{\mathcal{G}}(\{x_{i,h} \wedge l^u_{\nu_{i,j}}\}|\mathbf{Y} = \mathbf{y})$ for link $l^u_{\nu_{i,j}} = (\nu_{k,z}, x_{k,g}) \in$ *inc*$(\nu_{i,j})$ is needed to construct the expected counts in (15), and can be computed in $\mathcal{G}_{\mathbf{Y}=\mathbf{y}}$ as:

$$P^{\mathcal{G}}(\{x_{i,h} \wedge l^u_{\nu_{i,j}}\}|\mathbf{Y} = \mathbf{y}) = \frac{1}{P^{\mathcal{G}}(\mathbf{Y} = \mathbf{y})} \mathit{ifl}(\nu_{k,z}) \cdot p^{\nu_{k,z}}_{x_{k,g}} \cdot \frac{\pi^{\nu_{k,z}}_{x_{k,g}}}{\mathit{ofl}(\nu_{i,j})} \cdot p^{\nu_{i,j}}_{x_{i,h}} \cdot \pi^{\nu_{i,j}}_{x_{i,h}} \ . \quad (23)$$

5. Estimating the Mutual Information with Missing Data

The mutual information between two random variables X and Y is defined as:

$$I(X,Y) = \sum_{i=1}^{|R(X)|} \sum_{j=1}^{|R(Y)|} p(x_i, y_j) \log \frac{p(x_i, y_j)}{p(x_i)p(y_j)} \ . \quad (24)$$

As the joint distribution of X and Y is unknown, we need to estimate the mutual information from data. Assume we have a database $\mathbf{D}$ probably containing missing data. We require estimates for $\theta_{ij} = p(x_i, y_j)$, $\theta_{i.} = p(x_i)$ and $\theta_{.j} = p(y_j)$ for $i = 1, \ldots, |R(X)|$ and $j = 1, \ldots, |R(Y)|$. Actually, we only need to estimate θ_{ij}, since $\theta_{i.} = \sum_{j=1}^{|R(Y)|} \theta_{ij}$ and $\theta_{.j} = \sum_{i=1}^{|R(X)|} \theta_{ij}$.

Since $\mathbf{D}$ may contain missing data, we can use the EM algorithm to estimate the required parameters. The detailed procedure is given in Alg. 2. Notice that steps 5 and 9 in algorithm 2 correspond, respectively, to the E and M steps of algorithm EM.

The value E_{ij} computed in line 7 of Alg. 2 is the expected number of records in $\mathbf{D}$ where X takes its i-th value and Y takes its j-th value. It is computed by exploring all the records $\mathbf{d} \in \mathbf{D}$ and calculating, for each record, the probability $P\{X = x_i, Y = y_j | \mathbf{d}, \mathbf{\Theta}^n\}$. That is, we compute:

$$E\left[\#_{\mathbf{D}}(X = x_i, Y = y_j)|\mathbf{\Theta}^n\right] = \sum_{\mathbf{d}\in\mathbf{D}} P\{X = x_i, Y = y_j|\mathbf{d}, \mathbf{\Theta}^n\} \ . \quad (25)$$

The probability in Eq. (25) will be equal to 0 if the record has a value different to (x_i, y_j) and equal to 1 if the record is exactly equal to (x_i, y_j). If some of the cells in the record is missing, the probability is computed using the current estimates $\mathbf{\Theta}^n$.

Algorithm 2 EM for estimating the mutual information

```
 1: procedure EM_MutualInformation($\mathbf{D}$)
 2:   Let $\mathbf{\Theta}^0 = \{\theta_{ij},\ i = 1, \ldots, |R(X)|,\ j = 1, \ldots, |R(Y)|\}$ be a random parametrisation of $p(x, y)$.
 3:   $n \leftarrow 0$.
 4:   repeat
 5:     for all $i = 1, \ldots, |R(X)|$ do
 6:       for all $j = 1, \ldots, |R(Y)|$ do
 7:         $E_{ij} \leftarrow E\left[\#_{\mathbf{D}}(X = x_i, Y = y_j)|\mathbf{\Theta}^n\right]$
 8:     $\mathbf{\Theta}^{n+1} \leftarrow \emptyset$
 9:     for all $i = 1, \ldots, |R(X)|$ do
10:       for all $j = 1, \ldots, |R(Y)|$ do
11:         $\theta_{ij}^{n+1} \leftarrow \frac{E_{ij}}{\sum_{k=1}^{|R(X)|}\sum_{l=1}^{|R(Y)|} E_{kl}}$
12:         $\mathbf{\Theta}^{n+1} \leftarrow \mathbf{\Theta}^{n+1} \cup \{\theta_{ij}^{n+1}\}$
13:     $n \leftarrow n + 1$.
14:   until $L(\mathbf{D}|\mathbf{\Theta}^n) \leq L(\mathbf{D}|\mathbf{\Theta}^{n-1})$.
15:   Estimate $I(X, Y)$ as:
          $$\hat{I}(X, Y) = \sum_{i=1}^{|R(X)|}\sum_{j=1}^{|R(Y)|} \theta_{ij}^{n-1} \log \frac{\theta_{ij}^{n-1}}{\theta_{i\cdot}^{n-1}\theta_{\cdot j}^{n-1}} \ .$$
16:   return $\hat{I}(X, Y)$.
```

6. Experiments

In this section we investigate experimentally the performance of our proposed procedure to learning from incomplete data. More specifically, we set out to answer the following questions:

1. Can we justify the use of the computationally heavy EM-principle for learning PDG models in situations when more efficient approaches exists?
2. How does the quality of PDG models learned by Alg. 1 compare to the quality of BN models learned by existing conventional procedures?

In order to test Alg. 1 we have performed experiments over three synthetic databases. The databases have been generated from random PDG models over 10, 20 and 40 variables. We will refer to these models as rnd10, rnd20 and rnd40 respectively. The models were generated with the following restrictions:

1. The models contain a single connected component[1].
2. Variables are categorical with between 2 and 5 states.
3. The models contain moderate branching on both the variable tree level and in the graph structure.

[1]This means that multiple connected components like the example structure of Fig. 1 is not possible.

4. All parameters where initialised to random multivariate distributions, following the procedure of Caprile [3].

From each of the 3 PDG models, we constructed four databases containing 250, 500, 1000 and 2000 complete samples. For each of the 12 databases, we have considered different rates of missing values, ranging from 5% to 30%. For each rate of missing values we generated 50 databases from the original (complete) database by randomly erasing the value in a fraction of the cells according to the rate of missing values. The learning algorithm was then executed on each of the 50 databases measuring the quality of the learned model as the log-likelihood of a separate validation database containing 10000 complete samples.

As score function we used the S_λ function of equation (2) with λ adjusted according to the size of the database to give a tradeoff between size and likelihood equivalent to the one imposed by the BIC score[2]. Finally, in order to speed up the algorithm, we put a limit of 10 iterations in each parametric EM[3] and 100 iterations in structural EM (the loop of Alg. 1).

6.1. Initial Results

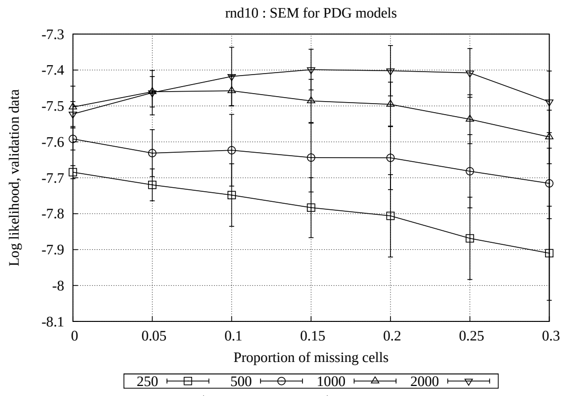
In Fig. 2(a-c) we show plots of mean and standard deviation of the log-likelihood of models learned in the experiments described above. That is, for each rate m of missing values and each sample size, 50 databases where generated by randomly removing m% of the values. The means and standard deviations were then estimated from these 50 results.

First, the plots of Fig. 2 in general show the expected behaviour as mean log-likelihood decreases as a result of increasing the proportion of missing cells in the training data, while standard deviation increases. We note, also as expected, that the experiments on the larger data sets reach higher likelihood on the validation data and also show a more stable performance with lower increase in standard deviation as the rate of missing values is increased.

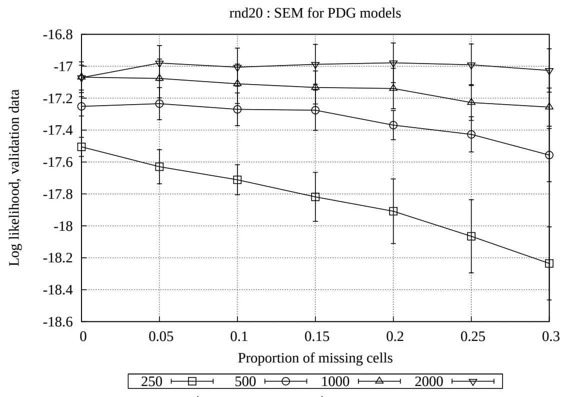
Second, in the experiment using 2000 samples from the rnd10 model (Fig. 2(a)) we observe an increase in likelihood up until a rate of 15% missing values. This behaviour may be caused by the algorithm over-fitting to the complete (0% missing values) training data, while the introduction of some missing values helps the algorithm learn a less specific model with better ability to generalise. That this over-fitting is most clearly pronounced for the larger databases may be explained by the fact that parameters where smoothed by adding a fictive count of 1 to every count. For smaller databases this will of course yield a more aggressive smoothing of parameters. We observe similar though less pronounced behaviour in the other two plots in Fig. 2(b-c).

[2]Setting $\lambda = \left(\frac{2N}{\log(N)} + 1\right)^{-1}$ where N is the number of observations yields BIC tradeoff.

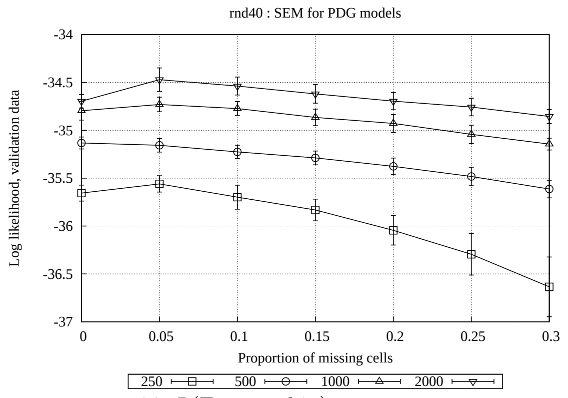
[3]We run a 100 iterations parametric EM to optimise the parameters of the final model.



(a) $L(\mathbf{D}_{\mathrm{val}}, \mathrm{rnd10}) = -6.85$



(b) $L(\mathbf{D}_{\mathrm{val}}, \mathrm{rnd20}) = -14.98$



(c) $L(\mathbf{D}_{\mathrm{val}}, \mathrm{rnd40}) = -32.10$

Figure 2: Mean and standard deviation of log-likelihood of a validation set of 10000 complete samples computed in the models learned from datasets sampled from model rnd10 (a), rnd20 (b) and rnd40 (c). The log-likelihood of the generating models are indicated beneath the plots.

6.1.1. The importance of the initial tree-structure

In this section we will investigate two aspects of the problem of choosing a starting point for our SEM algorithm. First, we study the effect of the structural learning applied after learning an initial tree from the Chow-Liu tree model. Next, we investigate the effect of using the generative models tree structure as a starting point.

The Chow-Liu tree model. As explained earlier, the initial tree structure is created using the classical algorithm of Chow and Liu [4] shown in Sect. 4. This initial model is itself a very commonly used model in probability estimation due to its simple restricted syntax and consequently efficient learning and inference. We therefore compare the quality of our final model to this initial model. From each experiment with missing data (72 total) we measured the likelihood of the validation data in the initial model as well as in the final PDG model. Using a Wilcoxon signed rank test for paired samples with significance level 0.05, we found significantly lower likelihood of the PDG model in only 2 cases, no significant difference in 5 cases while in 65 cases we found significant better likelihood of the PDG model.

The two cases in which the PDG model performed significantly worse was both using 500 samples of the rnd10 model, one with 25% and the other with 30% missing cells. This indicates that it is not a generally occurring phenomenon, but rather an observation that is specific to this one database when the rate of missing cells is high.

The 5 cases where no significant difference could be established was for the experiments using the 250 samples of rnd10 with 30% missing, the 500 samples of rnd10 with 20% missing, the 250 samples of rnd20 with 30% missing, the 500 samples of rnd20 with 30% missing and the 500 samples of rnd40 with 30% missing. The general property of these databases is a relatively small number of samples and a high degree of missing cells. This indicates that when data is limited and the degree of missing cells is high it has no significant effect on the model to try to optimise it by the local structural modifications.

As we do see a significant improvement in quality of the model in the 65 remaining experiments we will draw the conclusion in general (when data is not very limited and with a high degree of missing cells) it is worth the trouble of refining the induced Chow-Liu tree structure with split and merge operations.

The generating model as seed. The two phased learning procedure of first inducing a variable tree and the subsequently inducing a PDG model w.r.t. this tree raises the question of whether the trees learned by the Chow-Liu method is good starting point for the PDG learning. To gain insight into this question, we repeated the experiments with the synthetic data, but this time using the tree from the generating models in all experiments.

For each generative model (rnd10, rnd20 and rnd40) we have performed a Wilcoxon signed rank test for paired samples using the mean log-likelihood in the two measurements: 1) from learning a Chow-Liu tree as a starting point to the structural search and 2) using the trees extracted from the generative model as a starting point. The null hypothesis is that there are no difference, while the alternative hypothesis is that measurement 2 is higher than measurement 1, hence we are doing a one-sided test. The p-values are close to 1 for both rnd10 and rnd20, while for rnd40 we get a p-value of 6.45E-6. When inspecting the learned models in all three cases, we saw that the learned models is always much smaller than the generative model, which may indicate that the generative models contain much redundancy.

That the use of the tree structure of the generative model as a starting point for rnd10 and rnd20 does not result in significant improvement in the quality of the models indicates that for smaller to moderate sized domains, the underlying structure is less important. However, the low p-value for rnd40 indicates that for larger domains, the initial structure is more important to the final quality of the model.

6.1.2. Alternative approches to learning under the MCAR assumption

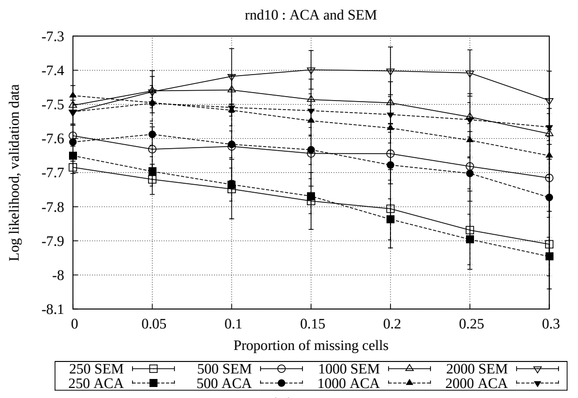
When data is missing completely at random (MCAR), there exist at least a few simple and valid alternatives to the EM-framework. We have experimented with two simple alternatives to EM, namely complete case analysis (CCA) and available case analysis (ACA). Both methods was briefly explained in Sec. 3. Here, we comment on experiments in which CCA (or ACA) was used for estimating parameters for a given structure (line 5 of Alg. 1).

As expected, CCA proved to be an inadequate alternative as the performance drops very fast and already performs significantly worse than SEM learning at the first level of 5% missing cells.

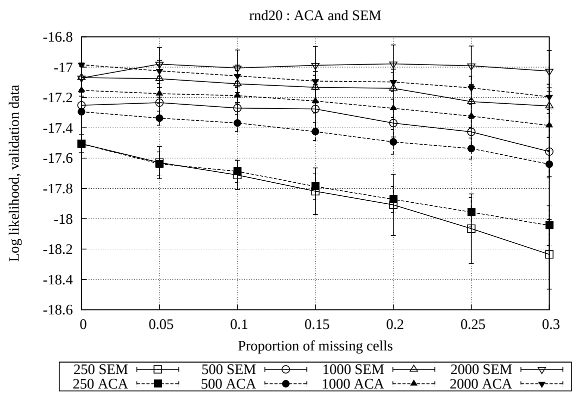
The performance of the more sophisticated ACA procedure is much more competitive. For each generative model (rnd10, rnd20 and rnd40) we performed a Wilcoxon signed rank test for paired samples using the mean log-likelihood in the two measurements: 1) from learning by SEM (Alg. 1) and 2) learning by ACA procedure. The null hypothesis is that there are no difference, while the alternative hypothesis is that measurement 1 is higher than measurement 2, hence we are doing a one-sided test. For none of the databases did we find significant support for stating that SEM performs better than ACA (p-value of 0.2257; 0.2759 and 0.1693, respectively). In Fig. 3 we include a plot showing the development of log-likelihood for each size of training data and rate of missing values. Here it can be seen that SEM usually performs better on larger training data, while ACA can beat SEM when training data is small. To answer the question of whether we can justify the use of the SEM algorithm compared to the conceptually simpler ACA procedure therefore seems to depend highly on the database, and no general rule can be stated.

6.2. PDG models vs. BN models

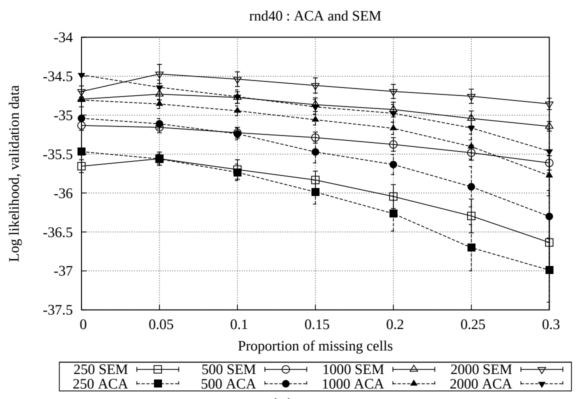
In this section we investigate the quality of the PDG models learned with the BN models learned by the SEM for BN models proposed by Friedman [6]. To this end, we have conducted an experimentation on the synthetic data described earlier as well as on two real databases.



(a)



(b)

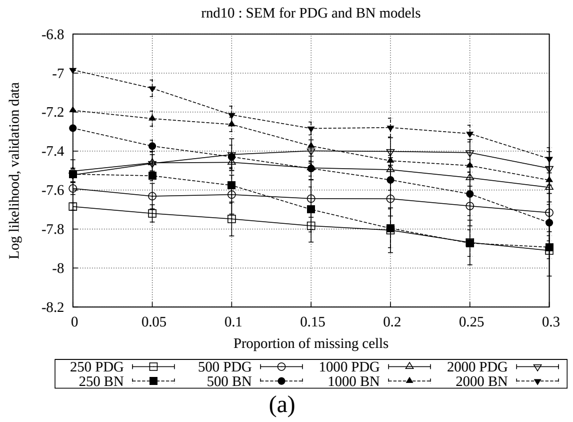


(c)

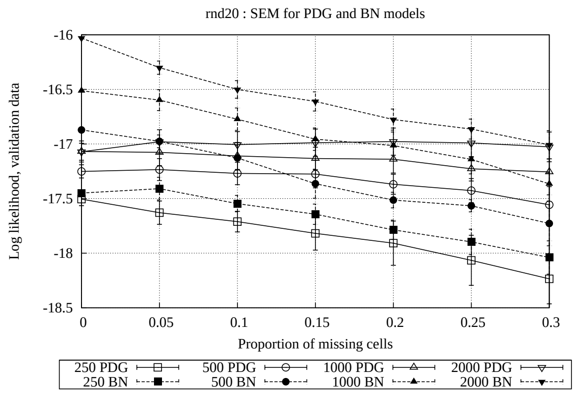
Figure 3: Comparison between a available case analysis (ACA) learning procedure and Algorithm 1 in plots (a-c).

6.2.1. Synthetic data

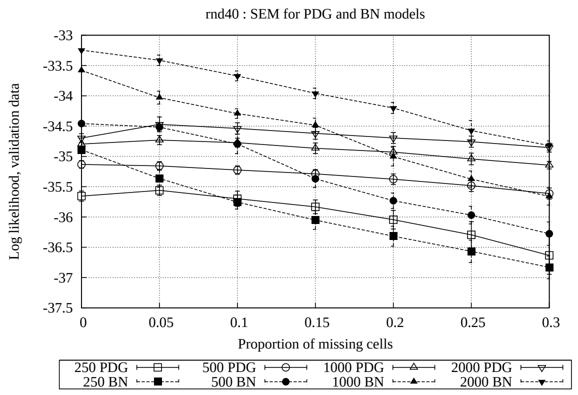
The results of the comparison between Alg. 1 for inducing PDGs and Friedman's structural EM for inducing Bayesian networks, for the synthetic databases



(a)



(b)



(c)

Figure 4: Comparison between the SEM of Friedman [6] for BN models and our SEM for PDG models (Alg. 1).

rnd10, rnd20 and rnd40 are displayed in Figure 4, where the average of the log-likelihood over all the records over a separate 10000-register test database is shown for database sizes ranging from 250 to 2000 and missing rates from 5% to 30%.

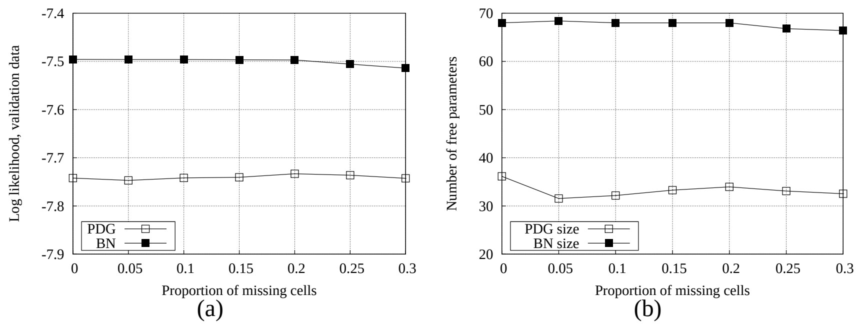


Figure 5: Comparison between the SEM of Friedman [6] for BN models and our SEM for PDG models (Alg. 1) on rnd14 generated data. Subfigure (a) compares obtained log-likelihoods while (b) shows the size in number of free parameters defined by each model respectively.

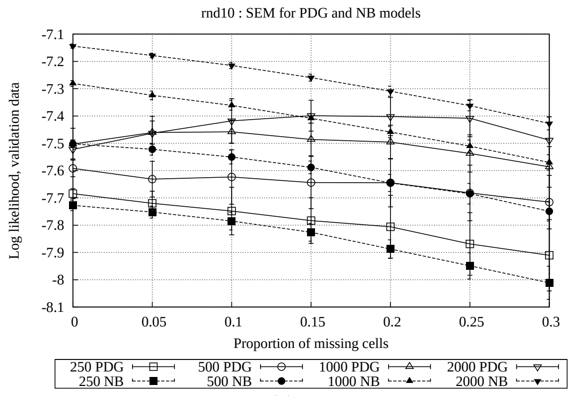
The target was to show if Alg. 1 is able to obtain PDGs comparable to BNs in terms of likelihood and size (number of parameters). It would result in an advantage in favour of PDGs if they are going to be used for probabilistic inference, as they are generally more efficient, in relation to size, for probabilistic inference tasks, than BNs [8]. The plots in Figure 4 agree with this target, but usually with an edge in favour of BN models in terms of likelihood, except for high missing rates, in which PDGs get much closer. In order to check whether the sizes of the obtained models also agreed with the expected results, we measured the number of parameters of the models learnt from each database, finding out that the BNs were much more compact in general than PDGs, especially for high missing rates. This fact seemed surprising. However, we noticed that the learnt BNs were usually sparse, with many disconnected variables. It suggests that the data used in the experiments actually contained many independencies, especially for small databases and high missing rates. It would explain that the obtained PDGs have more parameters than the BNs in general, because unlike the algorithm for learning from complete data presented in [9], we do not allow the variables to conform a forest, but just a single tree, due to the difficulties to carry out the χ^2 tests. This means that in order to represent in the structure a variable as independent from the rest, it would have to be represented by a single node, which in turn means that some merge operations should be performed. Our merge operation, as defined in Sec. 4.1 may be too restricted by the requirement that the children of two nodes being considered for a merge must be the same. Therefore, the PDGs obtained by Alg. 1 are prone to be more complex than necessary if there are many independencies supported by the data.

We conducted a second experiment to check this conjecture. The scheme was the same used for the former databases, but now we used another data set containing fewer independencies and larger size compared to the sample space. We obtained it by sampling 5000 records from a BN densely connected containing 14 binary variables. This network is a subnetwork of the random network used in [2]. The results of this experiment, in terms of likelihood measured on a separate test set of size 5000, are shown in Figure 5. The obtained models report similar likelihoods, again with a slight edge for BNs. However, in this case the PDGs are much more compact, with sizes around one half of the BNs. That PDGs are more compact is important when one wishes to perform exact inference as for PDGs this is possible in linear time in the model size, while the same is not true in general for BN models.

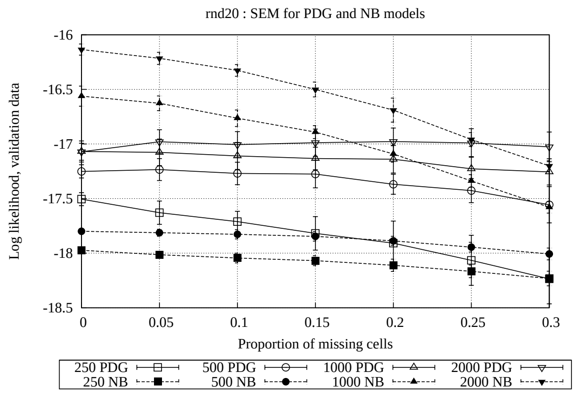
A more restricted version of the BN model is the *Naïve Bayes model* (NB) for probability estimation. This is a special kind of BN model in which the structure is fixed to include no connections between the observed variables and include an artificial latent variable that is the unique parent of every variable. The NB model has been shown to provide good accuracy when learned from complete data (see [11]), and we have performed experiments with this model to investigate its performance when learned from incomplete data. The NBE algorithm proposed by Lowd and Domingos [11] learns a NB model by iteratively increasing the cardinality of the latent variable and estimating parameters for the model by applying standard EM, until the likelihood of the model given a separate hold-out dataset did not increase. In our experiments we used a modified version of the NBE algorithm where the main difference was that we used the score of Eq. (2) with λ configured to tradeoff size and likelihood as the BIC score to determine convergence.

In Fig. 6 we have included plots showing detailed information on the learning of NB models plotted together with the log-likelihood. While the performance of NBs for data generated from rnd10 and rnd20 models (Fig. 6(a-b)) agrees with our corresponding observations for BNs (Fig. 4(a-b)), the picture changes for rnd40 (Fig. 6(c)). For the rnd40 data the NB models do not perform competitively when compared to PDG models. That NB models performs almost as competitively in the first two experiments is in accordance with the general findings for complete data as reported in [11]. As explained above, we use a score metric to determine convergence instead of monitoring likelihood over a separate hold-out dataset as was originally done for complete data in [11]. Using likelihood over a separate hold-out dataset instead of a score metric measured over training data, tackles more directly the problem of over-fitting. This may provide one explanation to our observation. One other possible explanation is that the EM-parameter estimation gets stuck in a local optima, which is a well known problem for the EM procedure. A typical NB model from the rnd40 experiments contains approximately 30000 free parameters while no PDG model contained more than 1000 free parameters. This observation is also important when considering computational complexity of exact inference, as both models provide exact inference in time linear in their size.

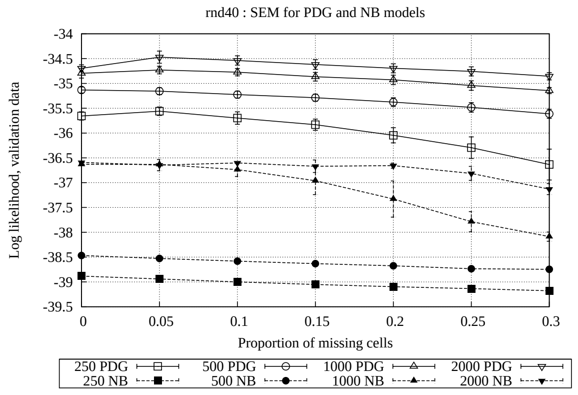
It should be noted that the NB model is a member of a broader class of models that share the property of allowing exact inference to be computed in time linear in the size of the model, namely the class of decomposable models. While it is known that for every decomposable model there exists an equivalent PDG model that has size at most linear to the size of the given decomposable model, the converse is not true in general (see [8]).



(a)



(b)



(c)

Figure 6: Comparison between SEM for PDG models (Alg. 1) and learning of NB models.

6.2.2. Real data

For the experiment with real data, we used two databases containing missing values, both publicly available from the UCI repository (see [13]).

house-votes-84 : This database contains information on the disposition of each of

		PDG	BN	NB
L	house-votes-84	-7.1256±3.0131	-7.0306±3.2524	-7.1143±2.9824
	soybean-large	-15.9726±4.7736	-14.1852±4.9756	-17.7534±5.2708
S	house-votes-84	49	152	2973
	soybean-large	1085	2091	13693

Table 1: Results of leave-one-out analysis of incomplete real data where the missing completely at random assumption may not be valid. Numbers in the 'L' rows are mean and standard deviation of log-likelihood from leave-one-out analysis. Numbers in the 'S' rows are the size of a model learned from the full database.

the 435 U.S. House of Representatives congressmen/-women on 16 key votes (for or against) and their party affiliation (democrat or republican) from the year 1984. In 288 cases the disposition of the specific congressman/-woman was unknown, and we treat this as a missing value.

soybean-large : This database contains 307 observations of soybean plants and the general health of the plants. For each plant up to 35 pieces of relevant information is recorded (eg. stem condition, condition of leafs etc.) and some auxiliary information (date, temperature) together with a label of one out of 15 diseases.

As the missing values are fixed in these databases, we followed a leave-one-out strategy to test the algorithms. The results are shown in Table 1. Once more, the likelihood is better for BNs, but the PDGs are much more compact, which suggests that these databases do not contain as many independencies as rnd10, rnd20 and rnd40.

The answer to the question of PDGs performance in comparison to existing BN-based approaches put forth in the beginning of this section would then be that it depends on the use of the recovered models. While unrestricted BN models and their structure is a rich tool for discussing dependencies found in the data, PDGs may provide for a more efficient way to do exact inference.

7. Conclusions

In this paper we have introduced an algorithm for learning PDG models in the presence of missing data. Our proposal was inspired by previous work on learning BN models from incomplete data by Friedman [6]. With this algorithm, we have extended the class of problems that can be approached using PDGs.

The experiments conducted show a reasonably good performance of the algorithm. First, a significant improvement over the initial Markov tree models was demonstrated. Second, the degrading effect on the performance as the rate of missing data increases is moderate. Third, the experiments carried out to compare the PDG models with simpler approaches to the missing completely at random problem, turned out mainly positive for the PDG learning. Available-case-analysis has the most competitive performance, and the preference of ACA method vs. our SEM algorithm seems to depend on the size of the training data available.

The comparison with the BNs obtained by Friedman's structural EM and the simpler NB learning is much more level. The BNs and NBs are usually slightly better in terms of likelihood, but the PDGs are typically smaller than both NBs and BNs, with the exception of BNs for problems where there are many independencies. However, it must be pointed out that PDGs are usually employed as tools for probabilistic inference, and in that case the efficiency in relation to the number of parameters is higher for PDGs [8].

The algorithm introduced here can be extended in various ways. For instance, the use of other scores could be considered. Also, a Bayesian approach could be followed as in [7]. Another aspect to be further studied is how to allow the induction of forests of variables instead of a single tree, with the aim of being able to get more compact models.

It should be noted that the approach presented in this paper does not guaranty to recover the generative model when such a model exists, even with a large data sample. In the case of complete data this is also still an open problem that is to be investigated in future studies. Another obviously interesting focus for future studies include the extension of the current algorithm to handle scenarios where unobserved (hidden) variables are known to influence the observed data.

Acknowledgements

This work has been supported by the Spanish Ministry of Education and Science, through projects TIN2007-67418-C03-01,02 by FEDER funds.

References

[1] Bozga, M. and Maler, O. (1999). On the representation of probabilities over structured domains. In *Proceedings of the 11th International Conference on Computer Aided Verification*, pages 261–273. Springer.

[2] Cano, A., Moral, S., and Salmerón, A. (2000). Penniless Propagation in Join Trees. *International Journal of Intelligent Systems*, 15(11):1027–1059.

[3] Caprile, B. (2001). Uniformly Generating Distribution Functions for Discrete Random Variables. Technical report, ITC-irst - Centro per la Ricerca Scientifica e Tecnologica, Italy.

[4] Chow, C. K. and Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467.

[5] Dempster, A. P., Laird, N. M., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

[6] Friedman, N. (1997). Learning belief networks in the presence of missing values and hidden variables. In *Proceedings of the Fourteenth International Conference on Machine Learning*.

[7] Friedman, N. (1998). The Bayesian structural EM algorithm. In *Proceedings of the UAI'98 Conference*.

[8] Jaeger, M. (2004). Probabilistic decision graphs - combining verification and AI techniques for probabilistic inference. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 12:19–42.

[9] Jaeger, M., Nielsen, J. D., and Silander, T. (2006). Learning probabilistic decision graphs. *International Journal of Approximate Reasoning*, 42(1-2):84–100.

[10] Lauritzen, S. L. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201.

[11] Lowd, D. and Domingos, P. (2005). Naïve Bayes Models for Probability Estimation. In *Proceedings of the Twentysecond International Conference on Machine Learning*, pages 529–536.

[12] McLachlan, G. J. and Krishnan, T. (1997). *The EM Algorithm and Extensions*. Wiley & Sons.

[13] Newman, D., Hettich, S., Blake, C., and Merz, C. (1998). UCI repository of machine learning databases: `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[14] Nielsen, J. D. and Jaeger, M. (2006). An empirical study of efficiency and accuracy of probabilistic graphical models. In *Proceedings of the Third European Workshop on Probabilistic Graphical Models*, pages 215–222.

[15] Nielsen, J. D., Rumí, R., and Salmerón, A. (2007). El clasificador grafo de decisión probabilístico. Presented at: XXX Congreso Nacional de Estadística e Investigación Operativa. `http://www.ual.es/~dalgaard/publications/seio07.pdf`.

[16] Nielsen, J. D., Rumí, R., and Salmerón, A. (2008). Supervised classification using probabilistic decision graphs. *Computational Statistics & Data Analysis*, In Press, Accepted Manuscript:–.

[17] Peña, J. M., Lozano, J. A., and Larrañaga, P. (2000). An improved Bayesian structural EM algorithm for learning Bayesian networks for clustering. *Pattern Recognition Letters*, 21:779–786.

[18] Ramoni, M. and Sebastiani, P. (1997). Learning Bayesian networks from incomplete databases. Technical Report KMI-TR-43, Knowledge Media Institute, The Open University.