Probabilistic relational model benchmark generation: Principle and application

Mouna Ben Ishak, Philippe Leray, Nahla Ben Amor

PRM Benchmark Generation: Principle and Application to PRM Structure Learning

Mouna Ben Ishak*, Philippe Leray and Nahla Ben Amor
LARODEC Laboratory, ISG, Université de Tunis, Tunisia
DUKe research group, LINA Laboratory UMR 6241, University of Nantes, France

Abstract

Probabilistic relational models (PRMs) extend Bayesian networks (BNs) to a relational data mining context. Even though a panoply of works have focused, separately, on Bayesian networks and relational databases random generation, no work has been identified for PRMs on that track. This paper provides an algorithmic approach allowing to generate random PRMs from scratch to cover the absence of generation process. The proposed method allows to generate PRMs as well as synthetic relational data from a randomly generated relational schema and a random set of probabilistic dependencies. This can be of interest for machine learning researchers to evaluate their proposals in a common framework, as for databases designers to evaluate the effectiveness of the components of a database management system.

1 Introduction

Data mining is the central step in knowledge discovery in databases. It relies on several research areas, notably, statistics, databases and machine learning. Several basic machine learning models have been involved including Bayesian networks (BNs) [28]. These methods have been developed for data in the traditional matrix form. However, due to the developments of communications and storage technologies, data about the real world is seldom of this form. The data can present a very large number of dimensions, with several different types of entities. Recently, there has been growing interest in extracting patterns from such data representation. Statistical relational learning (SRL) is an emerging area of machine learning that enables effective and robust reasoning about relational data structures [19]. Through this paper, we are particularly interested in probabilistic relational

*Corresponding author: Mouna Ben Ishak. E-mail: mouna.benishak@gmail.com

models (PRMs)[1] [23, 29] which represent a relational extension of Bayesian networks [28], where the probability model specification concerns classes of objects rather than simple attributes.

PRMs have prove their applicability in several areas (e.g., risk analysis, web page classification, recommender systems) [6, 13, 32] as they allow to minimize data preprocessing and the loss of significant information [30]. The use of PRMs implies their construction either by an expert or by applying learning algorithms in order to learn the model from some existing observational relational data. PRMs learning implies finding a graphical structure as well as a set of conditional probability distributions that fit the best way to the relational training data. The evaluation of the learning approaches is usually done using randomly generated data coming from either real known networks or randomly generated ones. However, neither the first nor the second are available in the literature.

Moreover, there is a growing interest from the database community to produce database benchmarks allowing to support and illustrate decision support systems (DSSs). For real-world business tasks, uncertainty is an unmissable aspect. So, benchmarks designed to support DSSs should consider this task.

In this paper, we propose an algorithmic approach allowing to generate random PRMs from scratch, then populate a database instance. The originality of this process is that it allows to generate synthetic relational data from a randomly generated relational schema and a random set of probabilistic dependencies. Since PRMs bring together two neighboring subfields of computer science, namely machine learning and database management, our process can be useful for both domains. It is imperative for statistical relational learning researchers to evaluate the effectiveness of their learning approaches. On the other hand, it can be of interest for database designers to evaluate the effectiveness of a database management system (DBMS) components. It allows to generate various relational schemas, from simple to complex ones, and to populate database tables with huge number of tuples derived from underlying probability distributions defined by the generated PRMs.

This paper presents an extend version of a preliminary work published in [2]. We give a detailed presentation of the proposed process. Then, we approve our implementation via a practical application by applying our generation process to evaluate PRM structure learning algorithms.

The remainder of this paper is structured as follows. Section 2 presents some useful theoretical concepts and definitions. Section 3 explains the prin-

[1]Neville and Jensen [27] use the term relational Bayesian network to refer to Bayesian networks that have been extended to model relational databases [23, 29] and use PRM in its more general sense to distinguish the family of probabilistic graphical models that are interested in extracting statistical patterns from relational models. In this paper we preserve the term PRM as used by [23, 29].

ciple of our contribution and details it. Section 4 provides a toy example that illustrates all the steps of our process from the random generation of PRM and database to their population. Section 5 goes through the implementation strategy. Section 6 concerns the application of our proposal to PRM structure learning. Finally, Section 7 concludes and outlines some perspectives.

2 Background

This section first provides a brief recall on Bayesian networks and relational model, then introduces PRMs.

2.1 Bayesian networks

Bayesian networks (BNs) [28] are directed acyclic graphs allowing to efficiently encode and manipulate probability distributions over high-dimensional spaces. Formally, they are defined as follows:

Definition 1 *A Bayesian network* $B = \langle \mathcal{G}, \Theta \rangle$ *is defined by:*

1) A graphical component (structure): a directed acyclic graph (DAG) $\mathcal{G} = (V, E)$*, where* V *is the set of vertices representing* n *discrete random variables* $\mathcal{A} = \{A_1, \ldots, A_n\}$*, and* E *is the set of directed edges corresponding to conditional dependence relationships among these variables.*

2) A numerical component (parameters): $\Theta = \{\Theta_1, \ldots, \Theta_n\}$ *where each* $\Theta_i = P(A_i | Pa(A_i))$ *denotes the conditional probability distribution (CPD) of each node* A_i *given its parents in* $\mathcal{G}$ *denoted by* $Pa(A_i)$*.*

Several approaches have been proposed to learn BNs from data [9]. The evaluation of these learning algorithms requires either the use of known networks or the use of a random generation process. The former allows to sample data and perform learning using this data in order to recover the initial gold standard net. The latter allows to generate synthetic BNs and data in order to provide a large number of possible models and to carry out experimentation while varying models from simple to complex ones.

Random Bayesian networks generation comes to provide a graph structure and parameters. Statnikov et al. [33] proposed an algorithmic approach to generate arbitrarily large BNs by tiling smaller real-world known networks. The complexity of the final model is controlled by the number of tiling and a connectivity parameter which determines the maximum number of connections between one node and the next tile. Some works have been devoted to the generation of synthetic networks but without guarantees

of that every allowed graph is produced with the same uniform probability [22]. In [21] the authors have proposed an approach allowing to generate uniformly distributed Bayesian networks using Markov chains, known as the PMMixed algorithm. Using this algorithm, constraints on generated nets can be added with relative ease such as constraints on nodes degree, maximum number of dependencies in the graph, etc. Once the DAG structure is generated, it is easy to construct a complete Bayesian network by randomly generating associated probability distributions by sampling either Uniform or Dirichlet distributions. Having the final BN, standard sampling method can be used to generate observational data such as forward sampling [20].

2.2 Relational model

The manner how the data is organized in a database depends on the chosen database model. The relational model is the most commonly used one and it represents the basis for the most large scale knowledge representation systems [12]. Formally, the relational representation can be defined as follows:

Definition 2 *The relational representation consists of*

- *A set of relations (or tables or classes)* $\mathcal{X} = \{X_1, \ldots, X_n\}$. *Each relation* X_i *consists of two parts:*
  - *The heading (relation schema) consists of a fixed set of attributes* $\mathcal{A}(X) = \{A_1, \ldots, A_k\}$. *Each attribute* A_i *is characterized by a name and a domain denoted* D_i.
  - *The body consists of a set of tuples (or records). Each tuple associates for each attribute* A_i *in the heading a value from its domain* D_i.
- *Each relation has a key (i.e., a unique identifier, a subset of the heading of a relation* X_i*.) and, eventually, a set of foreign key attributes (or* ***reference slots*** ρ*). A foreign key attribute is a field that points to a key field in another relation, called the referenced relation. The associated constraint is a* ***referential constraint****. A chain of such constraints constitutes a* ***referential path****. If a referential path from some relation to itself is found then it is called a* ***referential cycle****. Relation headings and constraints are described by a* ***relational schema*** $\mathcal{R}$.

Database designs involving referential cycles are usually contraindicated [10].

Let's note that when working in the conceptual level (i.e., entity-relationship model), relationships with cardinalities *many_to_many* values are allowed.

These latter disappear when creating the relational model and are substituted with *many_to_one* ones with the emergence of a new table (relation) substituting the relationship.

Usually the interaction with a relational database is ensured by specifying queries using structured language (SQL), which on their part use some specific operators to extract significant meaning such as **aggregators**. An aggregation function γ takes a multi-set of values of some ground type, and returns a summary of it. Some requests need to cross long reference paths, with some possible back and forth. They use composed slots to define functions from some objects to other ones to which they are indirectly related. We call this composition of slots a **slot chain** K. We call a slot chain single-valued when all the crossed reference slots end with a cardinality equal to 1. A slot chain is multi-valued if it contains at least one reference slot ending with cardinality equal to *many*. Multi-valued slot chains imply the use of aggregators.

Generally, database benchmarks are used to measure the performance of a database system. They still remains trendy [8, 1, 11] because of hardware and software development and also the huge amount of data provided due to the developments of communication technologies and the need of storage it and manage it in such a manner it can be used later. A database benchmark consists of several subtasks (e.g., generating the transaction workload, defining transaction logic, generating the database) [17].

Random database generation consists on creating the database schema, determining data distribution, generating it and loading all these components to the database system under test. Several propositions have been developed in this context. The main issue was how to provide a large number of records using some known distributions in order to be able to evaluate the system results. [4, 5]. In some research, known benchmarks [2] are used and the ultimate goal is only to generate a large dataset [18]. Nowadays, several software tools are available (e.g. *DbSchema* [3], *DataFiller* [4]). They allow to populate database instances knowing the relational schema structure. Records are then generated on the basis of this input by considering that the attributes are probabilistically independent which is not relevant when these benchmarks are used to evaluate decision support systems. The Transaction Processing Performance Council (TPC)[5] organization provides the TPC-DS[6] benchmark which has been designed to be suitable with real-world business tasks which are characterized by the analysis of huge amount of data. The TPC-DS schema models the sales and sales returns process for

[2]http://www.tpc.org
[3]http://www.dbschema.com/
[4]https://www.cri.ensmp.fr/people/coelho/datafiller.html
[5]http://www.tpc.org
[6]http://www.tpc.org/tpcds

an organization. TPC-DS provides tools to generate either the the data sets or the query sets for the benchmark. Nevertheless, uncertainty management stays a prominent challenge to provide better rational decision making.

2.3 Probabilistic relational models

Probabilistic relational models [16, 23, 29] are an extension of BNs in the relational context. They bring together the strengths of probabilistic graphical models and the relational presentation. Formally, they are defined as follows [16]:

Definition 3 *A Probabilistic Relational Model* Π *for a relational schema* $\mathcal{R}$ *is defined by:*

1) A qualitative dependency structure $\mathcal{S}$ *: for each class (relation)* $X \in \mathcal{X}$ *and each descriptive attribute* $A \in \mathcal{A}(X)$*, there is a set of parents* $Pa(X.A) = \{U_1, \ldots, U_l\}$ *that describes probabilistic dependencies. Each* U_i *has the form* $X.B$ *if it is a simple attribute in the same relation or* $\gamma(X.K.B)$*, where* K *is a slot chain and* γ *is an aggregation function.*

2) A quantitative component, a set of conditional probability distributions (CPDs), representing $P(X.A|Pa(X.A))$*.*

The PRM Π is a meta-model used to describe the overall behavior of a system. To perform probabilistic inference this model has to be instantiated. A PRM instance contains for each class of Π the set of objects involved by the system and relations that hold between them (i.e., tuples from the database instance which are interlinked). This structure is known as a relational skeleton σ_r [16].

Definition 4 *A relational skeleton* σ_r *of a relational schema is a partial specification of an instance of the schema. It specifies the set of objects* $\sigma_r(Xi)$ *for each class and the relations that hold between the objects. However, it leaves the values of the attributes unspecified.*

Given a relational skeleton, the PRM Π defines a distribution over the possible worlds consistent with σ_r through a ground Bayesian network [16].

Definition 5 *A Ground Bayesian Network (GBN) is defined given a PRM* Π *together with a relational skeleton* σ_r*. A GBN consists of:*

1) A qualitative component:

- *A node for every attribute of every object* $x \in \sigma_r(X), x.A$*.*

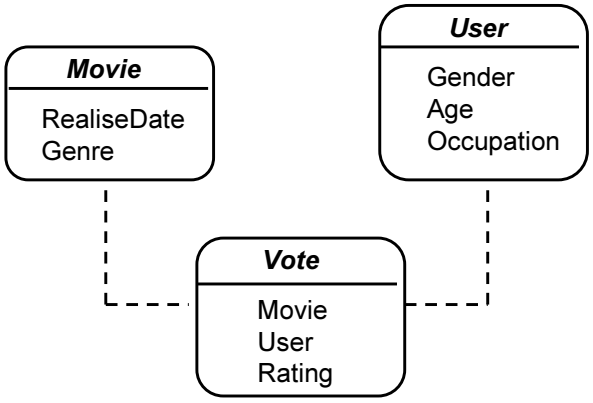

Figure 1: An example of relational schema

- *Each $x.A$ depends probabilistically on a set of parents $Pa(x.A) = u_1, \ldots u_l$ of the form $x.B$ or $x.K.B$, where each u_i is an instance of the U_i defined in the PRM. If K is not single-valued, then the parent is an aggregate computed from the set of random variables $\{y|y \in x.K\}, \gamma(x.K.B)$.*

2) A quantitative component, the CPD for $x.A$ is $P(X.A|Pa(X.A))$.

Example 1 *An example of a relational schema is depicted in Figure 1, with three classes $\mathcal{X} = \{Movie, Vote, User\}$. The relation $Vote$ has a descriptive attribute $Vote.Rating$ and two reference slots $Vote.User$ and $Vote.Movie$. $Vote.User$ relates the objects of class $Vote$ with the objects of class $User$. Dotted links presents reference slots. An example of a slot chain would be $Vote.User.User^{-1}.Movie$ which could be interpreted as all the votes of movies shown by a particular user.*
$Vote.Movie.genre \rightarrow Vote.rating$ is an example of a probabilistic dependency derived from a slot chain of length 1 where $Vote.Movie.genre$ is the parent and $Vote.rating$ is the child as shown by Figure 2. Also, varying the slot chain length may give rise to other dependencies. For instance, using a slot chain of length 3, we can have a probabilistic dependency from $\gamma(Vote.User.User^{-1}.Movie.genre)$ to $Vote.rating$. In this case, $Vote.rating$ depends probabilistically on an aggregate value of all the genres of movies voted by a particular user.

Figure 3 is an example of a relational skeleton of the relational schema of Figure 1. This relational skeleton contains 3 users, 5 movies and 9 votes. Also it specifies the relations between these objects, e.g., the user $U1$ voted for two movies $m1$ and $m2$.

Figure 4 presents the ground Bayesian network constructed from the relational skeleton of Figure 3 and the PRM of Figure 2. It resumes the same

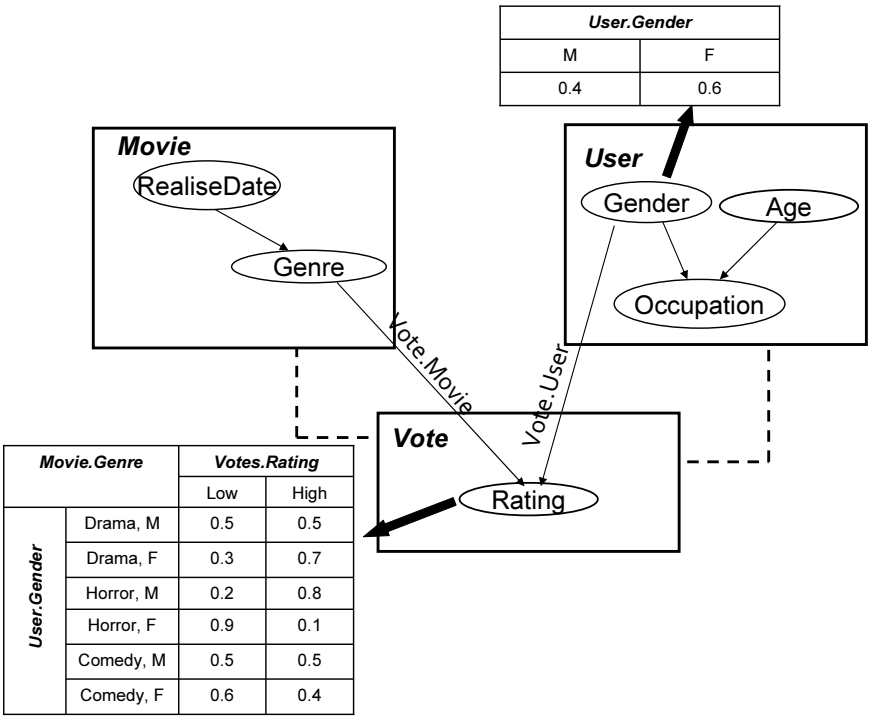

Figure 2: An example of probabilistic relational model

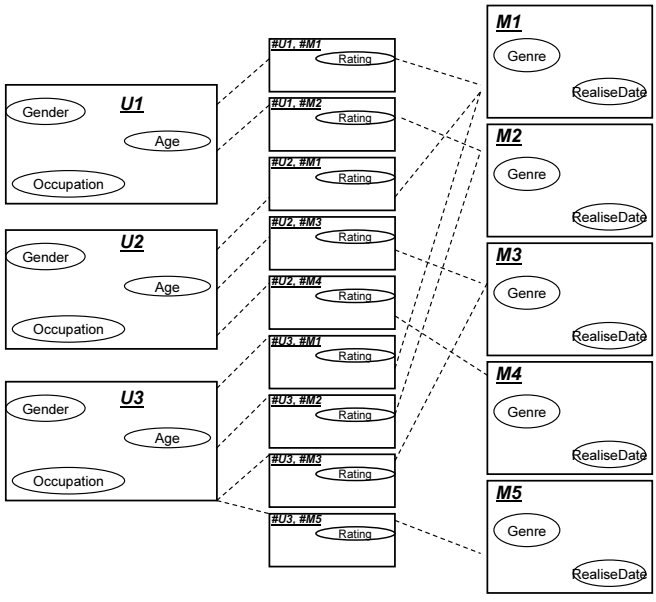

Figure 3: An example of a relational skeleton

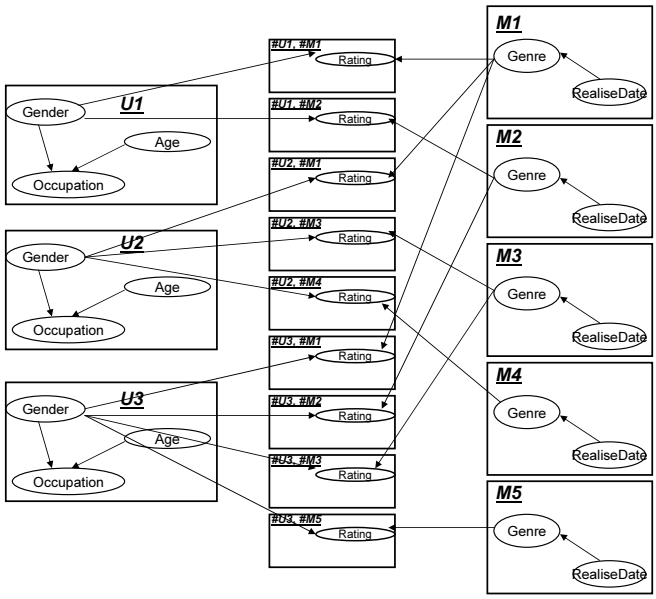

Figure 4: An example of a ground Bayesian network

dependencies as well as CPDs of the PRM at the level of objects. Here we omit to reproduce the CPDs to not overload the figure.

PRM structure learning has not been well studied in the literature. Only few works have been proposed to learn PRMs [14] or almost similar models [24, 25] from relational data.

Friedman et al. [14] proposed the Relational Greedy Hill-Climbing Search (RGS) algorithm. They used a greedy search procedure to explore the space of PRM structures while allowing increasingly large slot chains. PRM structures are generated using the *add_edge*, *delete_edge* and *reverse_edge* operators and aggregation functions if needed (cf. Section 2.3). As score function, they used a relational extension of the Bayesian Dirichlet (BD) [7] score expressed as follows:

$$\mathcal{RBD}_{score} = \sum_i \sum_{A \in \mathcal{A}(X_i)} \sum_{u \in V(Pa(X_i.A))} log[DM(\{C_{X_i.A}[v,u]\}, \{\alpha_{X_i.A}[v,u]\})] - \sum_i \sum_{A \in \mathcal{A}(X_i)} \sum_{u \in V(Pa(X_i.A))} length_{\mathcal{K}}(X_i.A, Pa(X_i.A)) \quad (1)$$

Where

$$DM(\{C_{X_i.A}[v,u]\}, \{\alpha_{X_i.A}[v,u]\}) = \frac{\Gamma(\sum_v \alpha[v])}{\Gamma(\sum_v (\alpha[v] + C[v]))} \prod_v \frac{\Gamma(\alpha[v] + C[v])}{\Gamma(\alpha[v])},$$

and

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

is the Gamma function.

As for standard BNs, evaluating the effectiveness of the proposed approaches is needed. However neither relational benchmarks nor general random generation process are available.

Random probabilistic relational models generation has to be established in order to evaluate proposed learning approaches in a common framework. [25] used a predefined schema and have only generated a number of dependencies varying from 5 to 15 and the conditional probability tables for attributes from a Dirichlet distribution. In [24] the authors have generated relational synthetic data to perform experimentation. Their generation process is based only on a particular family of relational schemas, with N classes (nodes) and $N-1$ referential constraints (edges). Referential constraints are then expressed using relationship classes witch gives rise to a final relational schema containing $2N-1$ relations. Whereas in real world cases, relational schemas may have more than $N-1$ referential constraints. If the schema is fully connected (as described in [26]), it will follows a tree structure. Torti et al. [34] proposed a slightly different representation of PRMs, developed in the basis of the object-oriented framework and expert knowledge. Their main issue is probabilistic inference rather than learning. In their experimental studies [35], they have randomly generated PRMs using the layer pattern. The use of this architecture pattern imposes a particular order when searching for connections between classes, generating reference slots of the relational schema and also when creating the relational skeleton. No indication has been made about the generation of probabilistic dependencies between attributes. In addition, they are not interested neither in populating a relational database nor in communicating with a database management system.

3 PRM Benchmark Generation

Due to the lack of famous PRMs in the literature, this paper proposes a synthetic approach to randomly generate probabilistic relational models from scratch and to randomly instantiate them and populate relational databases. To the best of our knowledge, this has not yet been addressed.

3.1 Principle

As we are working with a relational variety of Bayesian networks, then our generation process will be inspired from classical methods of random

Algorithm 1: RandomizePRM-DB

Input: N : the number of relations, $\mathcal{K}_{max}$: The maximum slot chain length allowed

Output: $\Pi :< \mathcal{R}, \mathcal{S}, CPD >$, $DB\_Instance$

begin

Step 1: Generate the PRM

$\Pi.\mathcal{R} \leftarrow Generate\_Relational\_Schema(N)$

$\Pi.\mathcal{S} \leftarrow Generate\_Dependency\_Structure(\Pi.\mathcal{R})$

$\Pi.\mathcal{S} \leftarrow Determinate\_Slot\_Chains(\Pi.\mathcal{R}, \Pi.\mathcal{S}, \mathcal{K}_{max})$

$\Pi.CPD \leftarrow Generate\_CPD(\Pi.\mathcal{S})$

Step 2: Instantiate the PRM

$\sigma_r \leftarrow Generate\_Relational\_Skeleton(\Pi.\mathcal{R})$

$GBN \leftarrow Create\_GBN(\Pi, \sigma_r)$

Step 3: Database population

$DB\_Instance \leftarrow Sampling(GBN)$

generation of BNs while respecting the relational domain representation.

The overall process is outlined in Algorithm 1. Roughly, the proposed generation process is divided into three main steps:

- The first step generates both the relational schema and the graph dependency structure using *Generate_Relational_Schema*, *Generate_Dependency_Structure* and *Determinate_Slot_Chains* functions respectively (Sections 3.2 and 3.3). Then the conditional probability tables are generated by the *Generate_CPD* function in the same way than Bayesian networks (cf. Section 2.1).

- The second step instantiates the model generated in the first step by generating the relational skeleton using the *Generate_Relational_Skeleton* function (Section 3.4). The *Create_GBN* function creates the GBN, from both the generated PRM and the generated relational skeleton, following the steps described in Section 2.3.

- The third step presents the *Sampling* function. It consists on database instance population and it may be performed using a standard sampling method over the GBN (Section 3.5).

Algorithm 2: *Generate_Relational_Schema*

Input: N : the number of classes
Output: $\mathcal{R}$: The generated relational schema

begin
  repeat
    $\mathcal{G} \leftarrow Generate\_DAG(Policy)$
  until *$\mathcal{G}$ is a connected DAG*;
  for *each relation* $X_i \in \mathcal{R}$ **do**
    $\mathcal{P}k\_X_i \leftarrow Genenerate\_Primary\_Key(X_i)$
    $\mathcal{A}(X_i) \leftarrow Genenerate\_Attributes(Policy)$
    $\mathcal{V}(X_i.A) \leftarrow Genenerate\_States(Policy)$
  for *each* $n_i \rightarrow n_j \in \mathcal{G}$ **do**
    $Fk\_X_i \leftarrow Generate\_Foreign\_Key(X_i, X_j, Pk\_X_j)$

3.2 Relational schema random generation

The relational schema generation process is depicted by Algorithm 2. Our aim is to generate, for a given number of classes (relations) N, a relational schema, with respect to the relational model definition presented in section 2.2 and where generated constraints allow to avoid referential cycles. We apply elements from the graph theory for random schema generation. We associate this issue to a DAG structure generation process, where nodes represent relations and edges represent Referential constraints definition. $X_i \rightarrow X_j$ means that X_i is the referencing relation and X_j is the referenced one. Besides, we aim to construct schemas where $\forall X_i, X_i \in \mathcal{X}$ there exist a referential path from X_i to X_j. This assumption allows to browse all classes in order to discover probabilistic dependencies later and it is traduced by searching DAG structures containing a single connected component (i.e., connected DAG).

Having a fixed number of relations N, the *Generate_DAG* function constructs a DAG structure $\mathcal{G}$ with N nodes, where each node $n_i \in \mathcal{G}$ corresponds to a relation $X_i \in \mathcal{R}$ following various possible implementation policies (cf. Section 5.2). For each class we generate a primary key attribute using the *Genenerate_Primary_Key* function. Then, we randomly generate the number of attributes and their associated domains using the *Genenerate_Attributes* and *Genenerate_States* functions respectively. Note that the generated domains do not take into account possible probabilistic dependencies between attributes. For each $n_i \rightarrow n_j \in \mathcal{G}$, we generate a foreign key attribute in X_i using the *Generate_Foreign_Key* function.

3.3 PRM random generation

Generated schemas are not sufficient to generate databases where the attributes are not independent. We need to randomly generate probabilistic dependencies between the attributes of the schema classes. These dependencies have to provide the DAG of the dependency structure $\mathcal{S}$ and a set of CPDs which define a PRM (cf. definition 3).

We especially focus on the random generation of the dependency structure. Once this latter is identified, conditional probability distributions may be sampled in a similar way as standard BNs parameter generation.

The dependency structure $\mathcal{S}$ should be a DAG to guarantee that each generated ground network is also a DAG [15]. $\mathcal{S}$ has the specificity that one descriptive attribute may be connected to another with different possible slot chains. Theoretically, the number of slot chains may be infinite. In practice a user-defined maximum slot chain length K_{max}, is specified to identify the horizon of all possible slot chains. In addition, the K_{max} value should be at least equal to $N-1$ in order to not neglect potential dependencies between attributes of classes connected via a long path. Each edge in the DAG has to be annotated to express from which slot chain this dependency is detected. We add dependencies following two steps. First we add oriented edges to the dependency structure while keeping a DAG structure. Then we identify the variable from which the dependency has been drawn by a random choice of a legal slot chain related to this dependency.

3.3.1 Constructing the DAG structure

The DAG structure identification is depicted by Algorithm 3. The idea here is to find for each node $X.A$ a set of parents from the same class or from further classes while promoting intra-class dependencies in order to control the final model complexity as discussed in [15]. This condition promotes the discovery of intraclass dependencies or those coming from short slot chains. More the chain slot is long, less a probabilistic dependency through this slot chain may be found. To follow this condition, having N classes, we propose to construct N separated sub-DAGs, each of which is built over attributes of its corresponding class using the *Generate_Sub_DAG* function. Then, we construct a super-DAG over all the previously constructed sub-DAGs. At this stage, the super-DAG contains N disconnected components: The idea is to add inter-classes dependencies in such a manner that we connect these disconnected components while keeping a global DAG structure.

To add inter-class dependencies we constrain the choice of adding dependencies among only variables that do not belong to the same class. For an attribute $X.A$, the *Generate_Super_DAG* function chooses randomly and an attribute $Y.B$, where $X \neq Y$, then verifies whether the super-DAG structure augmented by a new dependency from $X.A$ to $Y.B$ remains a DAG. If

Algorithm 3: Generate_Dependency_Structure

Input: $\mathcal{R}$: The relational schema
Output: $\mathcal{S}$: The generated relational dependency structure

begin
  for *each class* $X_i \in \mathcal{R}$ **do**
    $\mathcal{G}_i \leftarrow Generate\_Sub\_DAG(Policy)$
  $\mathcal{S} \leftarrow \bigcup \mathcal{G}_i$
  $\mathcal{S} \leftarrow Generate\_Super\_DAG(Policy)$

Algorithm 4: Determinate_Slot_Chains

Input: $\mathcal{R}$: The relational schema, $\mathcal{S}$: The dependency structure,
K_{max} : The maximum slot chain length
Output: $\mathcal{S}$: The generated relational dependency structure with generated slot chains

begin
  $K_{max} \leftarrow max(K_{max}, card(\mathcal{X}_{\mathcal{R}}) - 1)$
  for *each* $X.A \rightarrow Y.B \in \mathcal{S}$ **do**
    $Pot\_Slot\_Chains\_List \leftarrow$
    $Generate\_Potential\_Slot\_chains(X, Y, \mathcal{R}, K_{max})$
    for *each* $slot\_Chain \in Pot\_Slot\_Chains\_List$ **do**
      $l \leftarrow length(slot\_Chain)$
      $W[i] \leftarrow \exp^{\frac{-l}{nb\_Occ(l, Pot\_Slot\_Chains\_List)}}$
    $Slot\_Chain^* \leftarrow Draw(Pot\_Slot\_Chains\_List, W)$
    if $Needs\_Aggregator(Slot\_Chain^*)$ **then**
      $\gamma \leftarrow Random\_Choice\_Agg(list\_Aggregators)$
    if $Slot\_Chain^* = 0$ **then**
      $\mathcal{S}.Pa(X.A) \leftarrow \mathcal{S}.Pa(X.A) \cup Y.B$ *% here* $X = Y$
    else
      $\mathcal{S}.Pa(X.A) \leftarrow \mathcal{S}.Pa(X.A) \cup \gamma(Y.Slot\_Chain^*.B)$

yes it keeps the dependency otherwise it rejects it and searches for a new one. Used policies are discussed in Section 5.2.

3.3.2 Determining slot chains

During this step, we have to take into consideration that one variable may be reached through different slot chains and the dependency between two

descriptive attributes will depend on the chosen one. The choice has to be made randomly while penalizing long slot chains. We penalize long indirect slot chains, by having the probability of occurrence of a probabilistic dependence from a slot chain length l proportional to $\exp^{-l}$ [15]. Having a dependency $X.A \rightarrow Y.B$ between two descriptive attributes $X.A$ and $Y.B$, we start by generating the list of all possible slot chains ($Pot\_Slot\_Chains\_List$) of $length \leq K_{max}$ from which X can reach Y in the relational schema using the $Generate\_Potential\_Slot\_chains$ function. Then, we create a vector W of the probability of occurrence for each of the found slot chains, with $\log(W[i]) \propto \frac{-l}{nb\_Occ(l,Pot\_Slot\_Chains\_List)}$, where l is the slot chain length and nb_Occ is the number of slot chains of length $l \in Pot\_Slot\_Chains\_List$. This value will rapidly decrease when the value of l increases which allows to reduce the probability of selecting long slot chains. We then sample a slot chain from $Pot\_Slot\_Chains\_List$ following W using the $Draw$ function. If the chosen slot chain implies an aggregator, then we choose it randomly from the list of existing ones using the $Random\_Choice\_Agg$ function. The slot chain determination is depicted by Algorithm 4.

Following our approach, database population requires the instantiation of the previously generated PRM. Both steps are detailed below.

3.4 GBN generation

The generated schema together with the added probabilistic dependencies and generated parameters give rise to the probabilistic relational model. To instantiate this latter we need to generate a relational skeleton by generating a random number of objects per class, then adding links between objects. This step is strongly related to the reference slot notion. That is, all referencing classes have their generated objects related to objects from referenced classes. In Algorithm 5, we start by generating the number of objects of referencing and referenced classes using the $Draw\_Objects$ function. Then we generate the objects by randomly instantiating their corresponding classes using the $Generate\_Objects$ function. We specify the number of related ones using the $Draw\_Objects$ function and finally, we relate them using the Add_Links function. Used policies are discussed in Section 5.2.

The GBN is fully determined with this relational skeleton and the CPDs already present at the meta-level.

3.5 Database population

This process is equivalent to generating data from a Bayesian network. We can generate as many relational database instances as needed by sampling from the constructed GBN. The specificity of the generated tuples is that

Algorithm 5: Generate_Relational_Skeleton

Input: $\mathcal{R}$: The relational schema
Output: σ_r : The generated relational skeleton

begin
  for *each class* $\rho \in \mathcal{R}$ **do**
    $nb\_Objects_{\rho.referencing} \leftarrow Draw\_Objects(policy)$
    $nb\_Objects_{\rho.referenced} \leftarrow Draw\_Objects(policy)$
    $\mathcal{O}^{\rho.referenced} \leftarrow$
    $Generate\_Objects(nb\_Objects_{\rho.referenced}, \rho.referenced)$
    $\mathcal{O}^{\rho.referencing} \leftarrow$
    $Generate\_Objects(nb\_Objects_{\rho.referencing}, \rho.referencing)$
    $nb\_Related \leftarrow Draw\_Objects(policy)$
    $\sigma_r \leftarrow Add\_Links(\mathcal{O}^{\rho.referenced}, \mathcal{O}^{\rho.referencing}, nb\_Related)$

they are sampled not only from functional dependencies but also from probabilistic dependencies provided by the randomly generated PRM.

4 Toy example

In this section, we illustrate our proposal through toy example.

Relational schema generation. Figure 5 presents the result of running Algorithm 2, with $N = 4$ classes. For each class, a primary key has been added ($clazz0id$, $clazz1id$, $clazz2id$ and $clazz3id$). Then a number of attributes has been generated randomly together with a set of possible states for each attribute using the policies described in Section 5.2 (e.g., $clazz0$ has 3 descriptive attribues $att0, att1$ and $att2$. $att0$ is a binary variable). Finally, foreign key attributes have been specified following the DAG structure of the graph $\mathcal{G}$ (e.g., $clazz2$ references class $clazz1$ using foreign key attribute $clazz1fkatt12$).

PRM generation. We recall that this process consists of two steps: randomly generate the dependency structure $\mathcal{S}$ (Algorithm 3), then randomly generate the conditional probability distributions which is similar to parameter generation of a standard BN. The random generation of the $\mathcal{S}$ is performed in two phases. We start by constructing the DAG structure, the result of this phase is in Figure 6. Then, we fix a maximum slot chain length K_{max} to randomly determinate from which slot chain the dependency has been detected. We use $K_{max} = 3$, the result of this phase gives rise to the graph dependency structure of Figure 7. $\mathcal{S}$ contains 5 intra-class and 5 inter-class probabilistic dependencies.

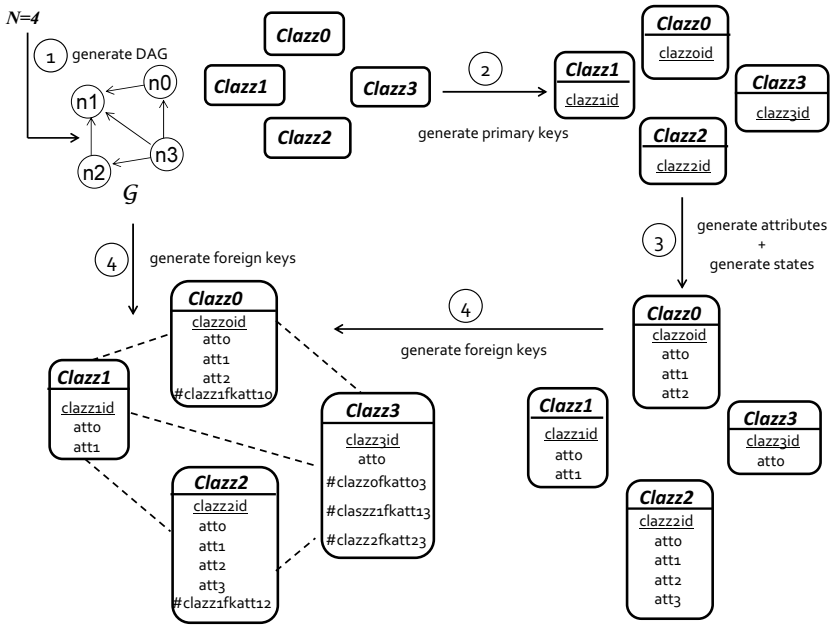

Figure 5: Relational schema generation steps

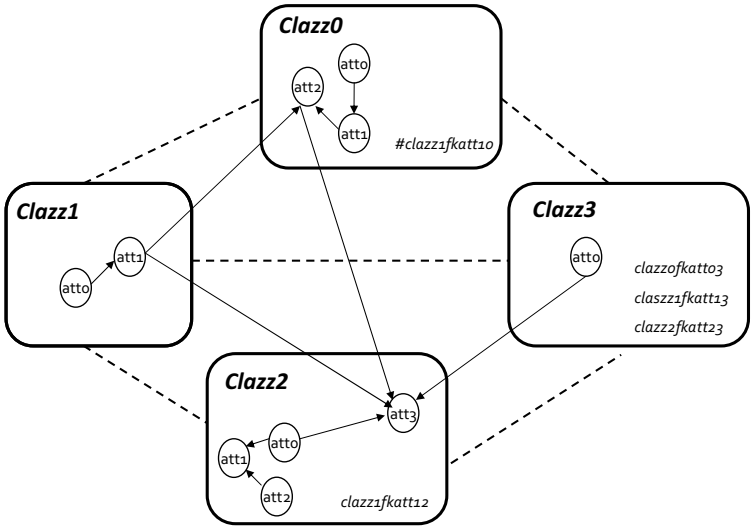

Figure 6: Graph dependency structure generation

Three of the inter-class dependencies have been generated from slot chains of length 1:
$Clazz0.clazz1fkatt10.att1 \rightarrow Clazz0.att2$;
$MODE(Calzz2.clazz2fkatt23^{-1}.att0) \rightarrow Clazz2.att3$ and;
$Clazz2.clazz1fkatt12.att1 \rightarrow Clazz2.att3$

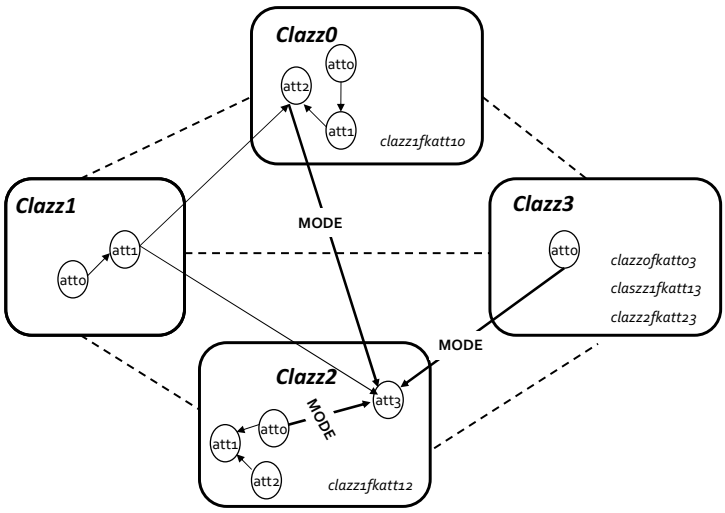

Figure 7: Example of a generated relational schema where the dotted lines represent referential constraints and the generated PRM dependency structure where the arrows represent probabilistic dependencies. we omit to specify slot chains to not overload the figure. Details about slot chains from which probabilistic dependencies have been detected are given in Paragraph *PRM generation*.

One from slot chain of length 2:
$MODE(Clazz2.clazz1fkatt12.clazz1fkatt12^{-1}.Clazz2.att0) \rightarrow Clazz2.att3$
One from slot chain of length 3:
$MODE(Calzz2.clazz2fkatt23^{-1}.claszz1fkatt13.clazz1fkatt10^{-1}) \rightarrow Clazz2.att3$

GBN creation. Once the PRM is generated, we follow the two steps presented in Section 3.4 to create a GBN and to populate the DB instance. We have generated an average number of 1000 tuple per class. The database has been stored using PostgreSQL [7] RDBMS. Figure 8 presents a graphical representation of the generated relational schema using SchemaSpy[8] software tool and some table records viewed from PostgreSQL interface.

5 Implementation

This section explains the implementation strategy of our generator, identifies the chosen policies and discusses the complexity of the algorithms.

[7] http://www.postgresql.org/
[8] http://schemaspy.sourceforge.net/

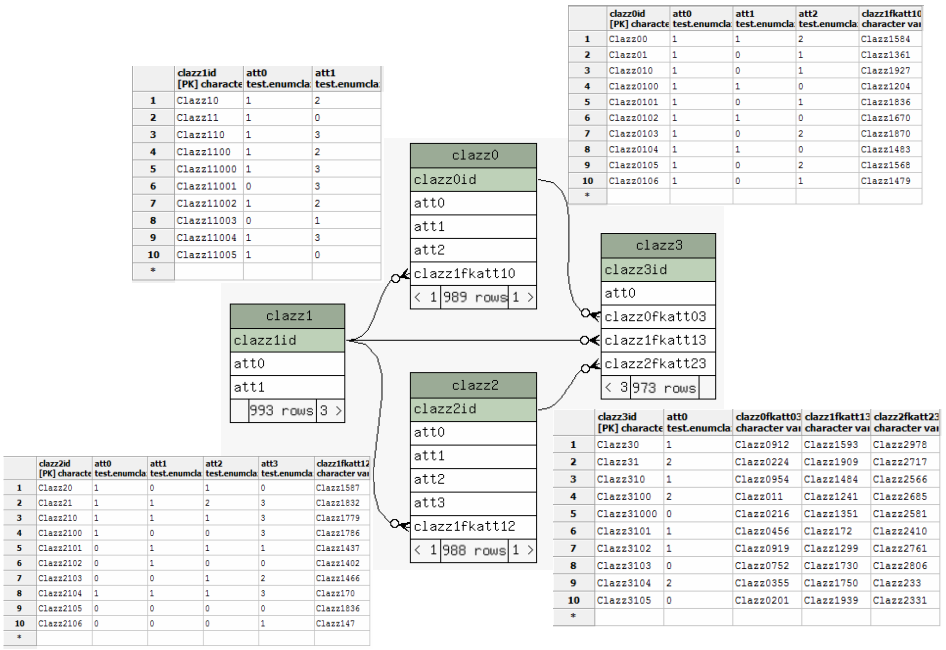

Figure 8: Visual graph representation of the generated relational schema and table records by using SchemaSpy and PostgreSQL software tools.

5.1 Software implementation

We implemented all the proposed algorithms into our local C++ PILGRIM API which is under development to provide an efficient tool to deal with several probabilistic graphical models (e.g., BNs, Dynamic BNs, PRMs). We use the Boost graph library[9] to manage graphs, the ProBT API[10] to manipulate BNs objects and the dtl library[11] to communicate with the PostgreSQL[12] RDBMS.

There is currently no formalization of PRMs, so we propose an enhanced version of the XML syntax of the *ProbModelXML*specification[13] to serialize our generated models. We added new tags to specify notions related to relational schema definition and we used the standard $\langle AdditionalProperties \rangle$ tags to add further notions related to PRMs (e.g., aggregators associated with dependencies, classes associated with nodes).

[9]http://www.boost.org/
[10]http://www.probayes.com/fr/Bayesian-Programming-Book/downloads/
[11]http://dtemplatelib.sourceforge.net/dtl_introduction.htm
[12]http://www.postgresql.org/
[13]http://www.cisiad.uned.es/techreports/ProbModelXML.pdf

5.2 Implemented policies

Policy for generating the relational schema DAG structure. To randomly generate the relational schema DAG structure, we use the PMMixed algorithm (cf. Section 2.1). This latter leads to generate uniformly distributed DAGs in the DAGs space. Consequently the generated structure may be a disconnected graph yet, we are in need of a DAG structure containing a single connected component. To preserve this condition together with the interest of generating uniformly distributed examples, we follow the *rejection sampling technique*. The idea is to generate a DAG following the PMMixed principle, if this DAG contains just one connected component, then it is accepted, otherwise it is rejected. We repeat these steps until generating a DAG structure satisfying our condition.

Policies for generating attributes and their cardinalities. Having the graphical structure, we continue by generating, for each relation R, a primary key attribute, a set of attributes $\mathcal{A}$, where $card(\mathcal{A}) - 1 \sim Poisson(\lambda = 1)$, to avoid empty sets, and for each attribute $A \in \mathcal{A}$, we specify a set of possible states $\mathcal{V}(A)$, where $card(\mathcal{V}(A)) - 2 \sim Poisson(\lambda = 1)$.

Policies for generating the dependency structure. We follow the PMMixed algorithm principle to construct a DAG structure inside each class. Then, in order to add inter-class dependencies, we use a modified version of the PMMixed algorithm where we constrain the choice of adding dependencies among only variables that do not belong to the same class.

Policy for generating the relational skeleton. The number of generated objects either for the referenced or the referencing classes follows a $Poisson(\lambda = N1)$ distribution and the number of interlinked objects follows a $Poisson(\lambda = N2)$ distribution. $N1$ and $N2$ are user-defined.

5.3 Complexity of the generation process

We have reported this work to this stage as it is closely related to the choice of the implementation policies. Let N be the number of relations (classes), we report the average complexity of each step of the generation process.

Complexity of the relational schema generation process. Algorithm 2 is structured of three loops. Namely, the most expensive one is the first loop dedicated for the DAG structure construction and uses the PMMixed algorithm. Time complexity of the PMMixed algorithm is $\mathcal{O}(N * \lg N)$. This algorithm is called until reaching the stop condition (i.e., a connected DAG). Let T be the average number of calls of the PMMixed algorithm. T is the ratio of the number of all connected DAG constructed from N nodes [31] to the number of all DAGs constructed from N nodes [3]. Time complexity of Algorithm 2 is is $\mathcal{O}(T * N * \lg N)$.

Complexity of the dependency structure generation process. As for Algorithm 2, the most expensive operation of Algorithm 3 is the

generation of the DAG structure inside each class $X_{i\in\{1\dots N\}} \in \mathcal{X}$. Through Algorithm 2, a set of attributes $\mathcal{A}(X_i)$ has been generated for each X_i. As $card(\mathcal{A}(X_i)) - 1 \sim Poisson(\lambda = 1)$, following Section 5.2, Then the average number of generated attributes for each class is $lambda = 1 + 1 = 2$. Then time complexity of the algorithm is $\mathcal{O}(N * 2 * \lg 2)$.

Complexity of the slot chains determination process. The most expensive operation of Algorithm 4 is the *Generate_Potential_Slot_chains* method. This latter explore recursively the relational schema graph in order to find all paths (i.e., slot chains) of length $k \in \{0 \dots K_{max}\}$. Time complexity of this method is $\mathcal{O}(N^{K_{max}})$.

Complexity of the relational skeleton generation process. In Algorithm 5, the *generate_Objects* method allows to generate a random number of objects per class. The average number of generated objects per class $\sim Poisson(\lambda = N1) = N1$. The average number of attributes for each object is equal to the average number of attributes at the class level which is equal to 2. Let E be the number of $\rho \in \mathcal{R}$, then time complexity of this method is $\mathcal{O}(N1 * 2 * E)$.

6 Application to PRM structure learning

As we said previously (cf. Section 2.3), only few works have been proposed to learn PRMs from data. In this section, we illustrate the utility of our contribution by applying it to evaluate the quality of a structure learning algorithm.

Network	**# classes**	**# variables**	**# edges**	**#states Min-Max**	**#parents Min-Max**
PRM_1	1	3	2	2-2	1-1
PRM_2	2	5	4	2-2	1-2
PRM_3	3	8	6	2-3	1-1
PRM_4	4	8	7	2-3	1-2
PRM_5	5	11	7	2-4	1-2

Table 1: Probabilistic relational models used in the evaluation study

6.1 Benchmarks

We have used our generation process to generate a set of theoretical PRMs (i.e., gold models), from which we have sampled a set of complete relational observational datasets. We have varied the number of classes N from 1 (i.e., the particular case where the PRM collapse to a standard Bayesian network) to 5. The maximum slot chain length is $K_{max} = N$. Details about all networks included in the study are given in Table 1. For each

of the described networks, we have randomly sampled 5 relational observational complete datasets with 100, 500, 1000, 2000 and 3000 instances as an average number of objects per class for each. Generated PRMs as well as datasets are available via this link `https://drive.google.com/folderview?id=0B160ZHpTs0CfUGI0Tmc3VXdJX1U&usp=sharing`.

6.2 Learning algorithm

We have re-implemented the Relational Greedy Hill-Climbing Search algorithm in our experimental platform using the RBD score (cf. Section2.3). During the learning process, the maximum slot chain length is fixed to $K_{max} = N$, where N is the number of classes.

6.3 Evaluation metrics

To evaluate the results of the learning process, we used the only metrics used in this context found in [24], namely, the Precision, Recall and F-score measures detailed below.

Precision: The ratio of the number of relevant dependencies retrieved to the total number of relevant and irrelevant dependencies retrieved in the learned PRM dependency structure $\mathcal{S}_{Learned}$. Relevant dependencies are those that are present in the true model.

$$\text{Precision} = \frac{\text{Number of relevant dependencies retrieved in } \mathcal{S}_{Learned}}{\text{Number of dependencies in } \mathcal{S}_{Learned}} \quad (2)$$

Recall: The ratio of the number of relevant dependencies retrieved to the total number of relevant dependencies int the true PRM dependency structure $\mathcal{S}_{True}$, which is generated using the random generation process.

$$\text{Recall} = \frac{\text{Number of relevant dependencies retrieved } \mathcal{S}_{Learned}}{\text{Number of dependencies in } \mathcal{S}_{True}} \quad (3)$$

F-score: Theoretically, a perfect structure learning algorithm should provide a precision and a recall of value one, whereas, usually, these two requirements are often contradictory: A learning approach that returns the list of all possible dependencies will provide a 100% recall. Alternatively, we can have a high value of the precision, not because of the good quality of the learning approach but because it provides a few number of learned dependencies. For instance a learning approach that is able to provide only one learned dependency given a true model with 10 dependencies will have a 100% precision, if this learned dependency is relevant, against a very low recall. Their harmonic mean, the F-measure, is then used to provide a weighted average of both precision and recall.

Network	Sample size	Precision	Recall	F-Measure
PRM_1	100	0.60	0.60	0.60
PRM_1	500	0.70	0.70	0.70
PRM_1	1000	0.80	0.80	0.80
PRM_1	2000	0.60	0.60	0.60
PRM_1	3000	0.60	0.60	0.60
PRM_2	100	0.83	0.35	0.48
PRM_2	500	0.52	0.50	0.51
PRM_2	1000	0.55	0.55	0.55
PRM_2	2000	0.75	0.75	0.75
PRM_2	3000	0.70	0.70	0.70
PRM_3	100	0.77	0.43	0.56
PRM_3	500	0.72	0.60	0.65
PRM_3	1000	0.51	0.60	0.55
PRM_3	2000	0.65	0.83	0.73
PRM_3	3000	0.58	0.72	0.64
PRM_4	100	0.68	0.48	0.57
PRM_4	500	0.88	0.91	0.89
PRM_4	1000	0.65	0.74	0.69
PRM_4	2000	0.69	0.83	0.75
PRM_4	3000	0.66	0.83	0.74
PRM_5	100	0.62	0.59	0.60
PRM_5	500	0.90	0.87	0.88
PRM_5	1000	0.72	0.70	0.71
PRM_5	2000	0.75	0.82	0.78
PRM_5	3000	0.71	0.74	0.72

Table 2: Evaluation results in term of Precision, Recall and F-score for RGS algorithm. Reported values are averages over 5 runs of the RGS algorithm, on 5 generated DB instances, for each RBN from Table 1 and each sample size.

$$\text{F-score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

6.4 Results and interpretation

A summary of the *Precision*, *Recall* and F-score results is presented in Table 2. We report the averages over 5 runs of the RGS algorithm on 5 generated DB instances, from the distribution of the networks of Table 1, for each sample size.

We note that the structure learning algorithm applied to our generated

relational training datasets provides high values of both precision and recall. Precision is between 51% and 88% regardless of the size of the true model generated and the size of the training set. Recall is between 35% and 91% and is less than 50% only for three values where the training set is of small size. the F-score is also high as it presents the average of both precision and recall.

This experimental study shows the utility of our contribution and its significance when evaluating the quality of any PRM structure learning algorithm. The RGS evaluation process has been made using several generated theoretical PRMs and various relational observational database instances, of different sizes, sampled from those PRMs. Achieving these results was not possible without our benchmark generation process.

7 Conclusion and perspectives

We have developed a process allowing to randomly generate probabilistic relational models and instantiate them to populate a relational database. The generated relational data is sampled from not only the functional dependencies of a relational schema but also from the probabilistic dependencies present in the PRM.

We have made our generated PRMs as well as data available to researchers who are working in this area (cf. Section 6.1). As we lack of PRM benchmarks, we believe that this can be a useful tool for evaluating new proposals related to PRMs learning from relational data. We are also about to distribute our software into GPL license.

In our ongoing work we are establishing a new approach to learn PRMs structure. We aim, particularly, to prove the effectiveness of our proposal by comparing it with already existing PRM learning approaches, in a common framework, using our random generation process.

Our process can more generally be used by other data mining methods as a probabilistic generative model allowing to randomly generated relational data. Moreover, it can be enriched by a test queries component to help database designers to evaluate the effectiveness of their RDBMS components.

References

[1] R. Angles, A. Prat-Pérez, D. Dominguez-Sal, and J-L. Larriba-Pey. Benchmarking database systems for social network applications. In *First International Workshop on Graph Data Management Experiences and Systems*, pages 1–7. ACM, 2013.

[2] M. Ben Ishak, P. Leray, and N. Ben Amor. Random generation and population of probabilistic relational models and databases. In *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2014)*, pages 756–763, 2014.

[3] E. A. Bender and R. W. Robinson. The asymptotic number of acyclic digraphs, ii. *J. Comb. Theory, Ser. B*, 44(3):363–369, 1988.

[4] D. Bitton, C. Turbyfill, and D. J. Dewitt. Benchmarking database systems: A systematic approach. In *Proceedings of the 9th International Conference on Very Large Data Bases*, pages 8–19. ACM, 1983.

[5] N. Bruno and S. Chaudhuri. Flexible database generators. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 1097–1107. ACM, 2005.

[6] R. Chulyadyo and P. Leray. A personalized recommender system from probabilistic relational model and users' preferences. In *Proceedings of the 18th Annual Conference on Knowledge-Based and Intelligent Information & Engineering Systems*, pages 1063–1072, 2014.

[7] G.F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[8] C. A. Curino, D. E. Difallah, A. Pavlo, and P. Cudre-Mauroux. Benchmarking oltp/web databases in the cloud: The oltp-bench framework. In *Proceedings of the Fourth International Workshop on Cloud Data Management*, pages 17–20. ACM, 2012.

[9] R. Daly, Q. Shen, and S. Aitken. Learning Bayesian networks: approaches and issues. *The Knowledge Engineering Review*, 26:99–157, 2011.

[10] C. J. Date. *The Relational Database Dictionary, Extended Edition.* Apress, New York, 2008.

[11] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudr-Mauroux. Oltp-bench: An extensible testbed for benchmarking relational databases. 7(4):277–288, 2013.

[12] S. Dzeroski and N. Lavrac, editors. *Relational Data Mining.* Springer New York Inc., New York, NY, USA, 2001.

[13] E. Fersini, E. Messina, and F. Archetti. Probabilistic relational models with relational uncertainty: An early study in web page classification. In *Proceedings of the International Joint Conference on Web Intelligence and Intelligent Agent Technology*, pages 139–142. IEEE Computer Society, 2009.

[14] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1300–1309, 1999.

[15] L. Getoor. *Learning statistical models from relational data.* PhD thesis, Stanford University, 2002.

[16] L. Getoor, D. Koller, N. Friedman, A. Pfeffer, and B. Taskar. *Probabilistic Relational Models, In Getoor, L., and Taskar, B., eds., Introduction to Statistical Relational Learning.* MA: MIT Press, Cambridge, 2007.

[17] J. Gray. *Benchmark Handbook: For Database and Transaction Processing Systems.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[18] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger. Quickly generating billion-record synthetic databases. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 243–252. ACM, 1994.

[19] D. Heckerman, C. Meek, and D. Koller. *Probabilistic entity-relationship models, PRMs, and plate models, In Getoor, L., and Taskar, B., eds., Introduction to Statistical Relational Learning.* MA: MIT Press, Cambridge, 2007.

[20] M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Proceedings of Uncertainty in Artificial Intelligence 2 Annual Conference on Uncertainty in Artificial Intelligence (UAI-86)*, pages 149–163, Amsterdam, NL, 1986. Elsevier Science.

[21] J. S. Ide and F. G. Cozman. Random generation of Bayesian networks. In *Brazilian symp.on artificial intelligence*, pages 366–375. Springer-Verlag, 2002.

[22] J. S. Ide, F. G. Cozman, and F. T. Ramos. Generating random Bayesian networks with constraints on induced width. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence*, pages 323–327, 2004.

[23] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proc. AAAI*, pages 580–587. AAAI Press, 1998.

[24] M. Maier, K. Marazopoulou, D. Arbour, and D. Jensen. A sound and complete algorithm for learning causal models from relational data. In *Proceedings of the Twenty-ninth Conference on Uncertainty in Artificial Intelligence*, pages 371–380, 2013.

[25] M. Maier, B. Taylor, H. Oktay, and D. Jensen. Learning causal models of relational domains. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 531–538, 2010.

[26] M. E. Maier, K. Marazopoulou, and D. Jensen. Reasoning about independence in probabilistic models of relational data. *CoRR*, abs/1302.4381, 2013.

[27] J. Neville and D. Jensen. Relational dependency networks. *Journal of Machine Learning Research*, 8:653–692, 2007.

[28] J. Pearl. *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann, San Franciscos, 1988.

[29] A. J. Pfeffer. *Probabilistic Reasoning for Complex Systems*. PhD thesis, Stanford University, 2000.

[30] L. De Raedt. Attribute-value learning versus inductive logic programming: the missing links. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 1–8, 1998.

[31] R. W. Robinson. *Counting unlabeled acyclic digraphs, In C. H. C. LITTLE, Ed., Combinatorial Mathematics V, volume 622 of Lecture Notes in Mathematics*. Springer, Berlin / Heidelberg, 1977.

[32] T. Sommestad, M. Ekstedt, and P. Johnson. A probabilistic relational model for security risk analysis. *Computers & Security*, 29:659–679, 2010.

[33] A. R. Statnikov, I. Tsamardinos, and C. Aliferis. An algorithm for generation of large Bayesian networks. Technical report, Department of Biomedical Informatics, Discovery Systems Laboratory, Vanderbilt University, 2003.

[34] L. Torti, P. H. Wuillemin, and C. Gonzales. Reinforcing the object-oriented aspect of probabilistic relational models. In *Proceedings of the 5th Probabilistic Graphical Models*, pages 273–280, 2010.

[35] P. H. Wuillemin and L. Torti. Structured probabilistic inference. *Int. J. Approx. Reasoning*, 53(7):946–968, 2012.