Probabilistic Relational Model Benchmark Generation

Mouna Ben Ishak, Rajani Chulyadyo, Philippe Leray

Probabilistic Relational Model Benchmark Generation

Mouna Ben Ishak, Rajani Chulyadyo, and Philippe Leray

LARODEC Laboratory, ISG, Université de Tunis, Tunisia
DUKe research group, LINA Laboratory UMR 6241, University of Nantes, France
DataForPeople, Nantes, France

Abstract

The validation of any database mining methodology goes through an evaluation process where benchmarks availability is essential. In this paper, we aim to randomly generate relational database benchmarks that allow to check probabilistic dependencies among the attributes. We are particularly interested in Probabilistic Relational Models (PRMs), which extend Bayesian Networks (BNs) to a relational data mining context and enable effective and robust reasoning over relational data. Even though a panoply of works have focused, separately, on the generation of random Bayesian networks and relational databases, no work has been identified for PRMs on that track. This paper provides an algorithmic approach for generating random PRMs from scratch to fill this gap. 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 not only for machine learning researchers to evaluate their proposals in a common framework, but also for databases designers to evaluate the effectiveness of the components of a database management system.

Keywords: Probabilistic Relational Model, Relational data representation, Benchmark generation

1 Introduction

Data mining is the central step in knowledge discovery in databases. It relies on several research areas including statistics and machine learning. Usually, machine learning techniques are developed around flat data representation (i.e., matrix form) and are known as propositional learning approaches. However, due to the development of communication and storage technologies, data management practices have taken further aspects. Data

can present a very large number of dimensions, with several different types of entities. With the growing interest in extracting patterns from such data representation, relational data mining approaches have emerged with the interest of finding patterns in a given relational database [11] and Statistical Relational Learning (SRL) has emerged as an area of machine learning that enables effective and robust reasoning about relational data structures [18]. In this paper, we are particularly interested in Probabilistic Relational Models (PRMs)[1] [22, 28], which represent a relational extension of Bayesian networks [27], where the probability model specification concerns classes of objects rather than simple attributes. PRMs present a probabilistic graphical formalism that enables flexible modeling of complex relational interactions.

PRMs have proved their applicability in several areas (e.g., risk analysis, web page classification, recommender systems) [7, 12, 32] as they allow to minimize data preprocessing and the loss of significant information [30]. The use of PRMs implies their construction either by experts or by applying learning algorithms in order to learn the model from some existing observational relational data. PRMs learning involves finding a graphical structure as well as a set of conditional probability distributions that best fit 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 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 that allows to generate random PRMs from scratch, and 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

[1]Neville and Jensen [26] use the term 'Relational Bayesian Network' to refer to Bayesian networks that have been extended to model relational databases [22, 28] and use the term '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 [22, 28].

generated PRMs. This paper presents an extended version of a preliminary work published in [1, 2].

2 Background

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

2.1 Bayesian networks

Bayesian networks (BNs) [27] 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 any guarantee that every allowed graph is produced with the same uniform probability [21]. In [20], the authors have proposed an approach, called PMMixed algorithm, that allows the generation of uniformly distributed Bayesian networks using Markov chains. 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, such as forward sampling [19], can be used to generate observational data.

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 [11]. 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 *has two parts:*
  - *The heading (relation schema): 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: 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, possibly, 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***[2]*. Relation headings and constraints are described by a* ***relational schema*** $\mathcal{R}$.

Usually the interaction with a relational database is ensured by specifying queries using structured query 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

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

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. A database benchmark includes several subtasks (e.g., generating the transaction workload, defining transaction logic, generating the database) [16].

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, 6]. In some research, known benchmarks [3] are used and the ultimate goal is only to generate a large dataset [17]. Nowadays, several software tools are available (e.g., *DbSchema*[4], *DataFiller*[5]) 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)[6] organization provides the TPC-DS[7] 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 sales and the sales returns process for an organization. TPC-DS provides tools to generate either data sets or 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 [15, 22, 28] 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 [15]:

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*

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

$Pa(X.A) = \{U_1, \ldots, U_l\}$ *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 in the system and the 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 [15].

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 [15].

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* $x \in \sigma_r(X), x.A$*.*
- *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 cast 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 a parent of* $Vote.rating$ *as shown in Figure 2. Also, varying the slot chain length*

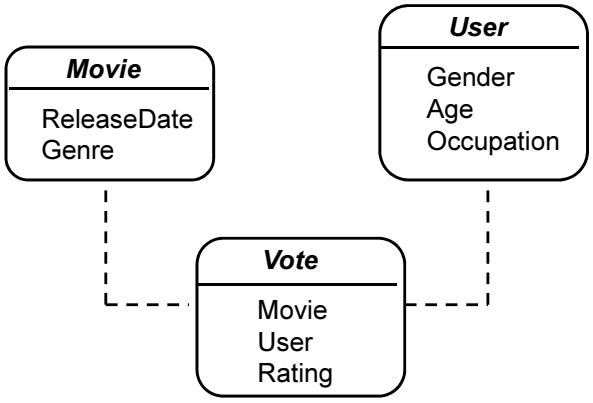
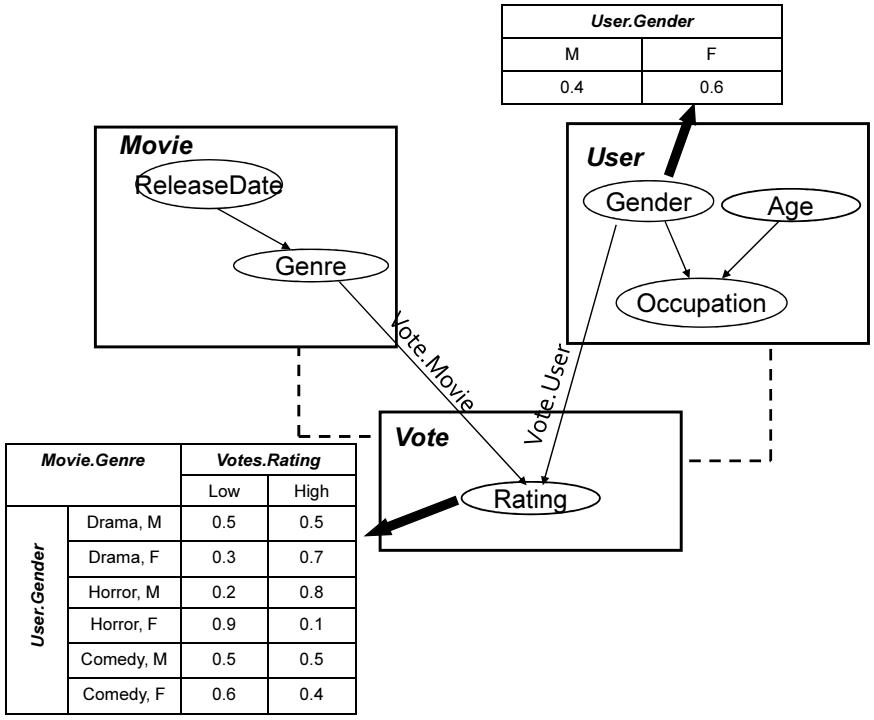


Figure 1: An example of relational schema



Movie.Genre		Votes.Rating	
		Low	High
User.Gender	Drama, M	0.5	0.5
	Drama, F	0.3	0.7
	Horror, M	0.2	0.8
	Horror, F	0.9	0.1
	Comedy, M	0.5	0.5
	Comedy, F	0.6	0.4

Figure 2: An example of probabilistic relational model

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.*

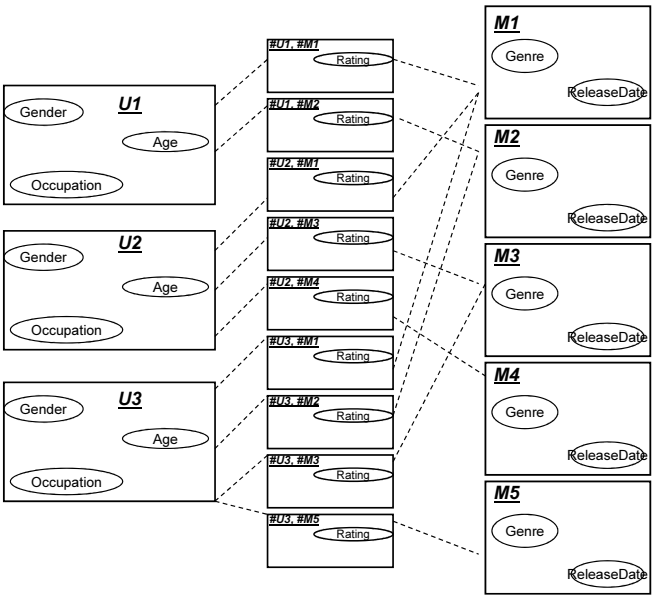


Figure 3: An example of a relational skeleton

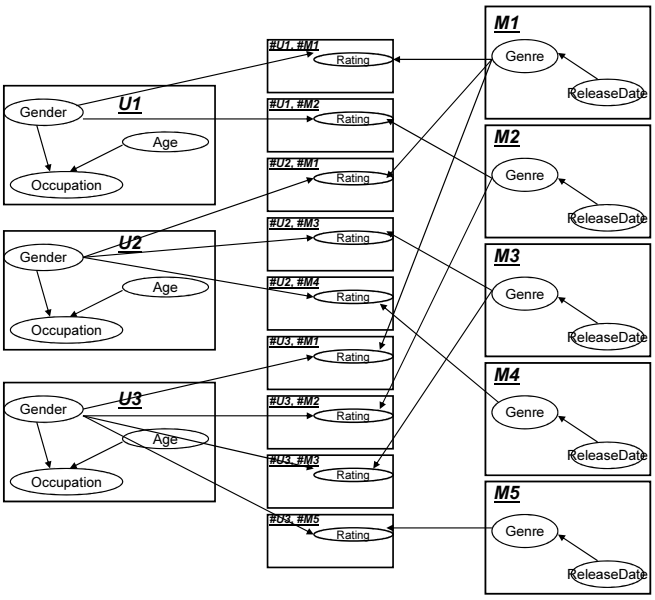


Figure 4: An example of a ground Bayesian network

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

dependencies as well as CPDs of the PRM at the level of objects. Here, we have not reproduced 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 [13] or almost similar models [23, 24] from relational data.

Friedman et al. [13] proposed Relational Greedy Hill-Climbing Search (RGS) algorithm. They applied 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 for score function, they used a relational extension of the Bayesian Dirichlet (BD) [8] score expressed as follows:

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

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. [24] 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 [23], 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. This 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 [25]), it will have 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 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, our generation process is inspired from classical methods of generation of random 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 a random PRM. For this, a relational schema is generated using *Generate_Relational_Schema* function(Section 3.2). Then, a graph dependency structure is generated using *Generate_Dependency_Structure* and *Determine_Slot_Chains* functions (Section 3.3). And finally, 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. First, a relational skeleton is generated using *Generate_Relational_Skeleton* function (Section 3.4). Then, using *Create_GBN* function, a ground Bayesian Network is generated from both the generated PRM and the generated relational skeleton.

- The third step presents the *Sampling* function. It involves database instance population and can be performed using a standard sampling

Algorithm 1: Generate_Random_PRM-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

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

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

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

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

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

$DB\_Instance \leftarrow Sampling(GBN)$

method over the GBN (Section 3.5).

3.2 Generation of a random relational schema

The relational schema generation process is depicted in Algorithm 2. Our aim is to generate a relational schema with a given number of classes such that it does not contain any referential cycles and also respects the relational model definition presented in section 2.2. We apply concepts 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 exists 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 $Generate\_Primary\_Key$ function. Then, we randomly generate the number of attributes and their associated domains using $Generate\_Attributes$

Algorithm 2: *Generate_Relational_Schema*

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

```
begin
    repeat
        G ← Generate_DAG(Policy)
    until G is a connected DAG;
    for each relation X_i ∈ R do
        Pk_X_i ← Generate_Primary_Key(X_i)
        A(X_i) ← Generate_Attributes(Policy)
        V(X_i.A) ← Generate_States(Policy)
    for each n_i → n_j ∈ G do
        Fk_X_i ← Generate_Foreign_Key(X_i, X_j, Pk_X_j)
```

and *Generate_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. Foreign key generation is limited to one attribute as foreign keys reference simple primary keys (i.e., primary keys generated from only one attribute).

3.3 Generation of a random PRM

Relational schemas are not sufficient to generate databases when the attributes are not independent. We need to randomly generate probabilistic dependencies between the attributes of the classes in the schema. 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 [14]. $\mathcal{S}$ 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

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**
    $\mathcal{G}_i \leftarrow Generate\_Sub\_DAG(Policy)$
  $\mathcal{S} \leftarrow \bigcup \mathcal{G}_i$
  $\mathcal{S} \leftarrow Generate\_Super\_DAG(Policy)$

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 presented in 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 [14]. This condition promotes the discovery of intra-class dependencies or those coming from short slot chains. The longer the slot chain, the lower is the chance of finding a probabilistic dependency through the slot chain. 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-class 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 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 so, it keeps the dependency otherwise it rejects it and searches for a new one. The policies that are used are discussed in Section 5.2.

Algorithm 4: Determine_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

$\mathcal{S}.Pa(X.A) \leftarrow \mathcal{S}.Pa(X.A) \cup \gamma(Y.Slot\_Chain^*.B)$

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. Following [14], the generation process has to give more priority to shorter slot chains for selection. Consequently, we have used the penalization term discussed in [14]. Longer indirect slot chains are penalized by having the probability of occurrence of a probabilistic dependency from a slot chain length l inversely proportional to $\exp^l$.

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 slot chains found, 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 in Algorithm 4.

Simplifying slot chains. While finding slot chains, duplicate slot chains might be encountered. By 'duplicate', we mean the slot chains which produce the same result. For example, in the schema of figure 1, $Vote.User$ and $Vote.User.User^{-1}.User$ are equivalent because traversing through the slot chains, we obtain the same set of User objects. Similarly, $Vote.Movie^{-1}.Movie$ is the same as an empty slot chain because this slot chain results in the target Movie object. When such duplicates are found, we pick the shorter one to avoid redundant, unnecessary computations. This is an improvement to our previous work [1, 2], where simplification of slot chains had not been considered. We apply the following rule to simplify slot chains.

A slot chain is represented as a sequence of reference slots and inverse slots as $\rho_1.\rho_2.\ldots.\rho_{n-1}.\rho_n$. If ρ_{n-1} is an inverse slot and $\rho_n^{-1} = \rho_{n-1}$, then the slot chain can be simplified by eliminating the last two slots. So, the simplified slot chain would, then, be $\rho_1.\rho_2.\ldots.\rho_{n-3}.\rho_{n-2}$. This can be done repetitively until no simplification is possible.

3.4 GBN generation

The generated schema together with the added probabilistic dependencies and generated parameters results in a probabilistic relational model. To instantiate this latter, we need to generate a relational skeleton. The GBN is, then, fully determined with this relational skeleton and the CPDs already present at the meta-level.

A relational skeleton can be imagined as a DAG where nodes are objects of different classes present in the associated relational schema and edges are directed from one object to another conforming to the reference slots present in the relational schema. This graph is, in fact, a special case of k-partite graph[8] of definition 6.

Definition 6 *Relational skeleton as a k-partite graph*

A relational skeleton is a special case of k-partite graph, $G_k = (V_k, E_k)$, with the following properties:

1. *The graph is acyclic,*

[8] A k-partite graph is a graph whose vertices can be partitioned into k disjoint sets so that there is no edge between any two vertices within the same set.

2. *All edges are directed (an edge $u \to v$ indicates that the object u refers to v, i.e., u has a foreign key which refers to the primary key of v),*

3. *Edges between two different types of objects are always oriented in the same direction, i.e. for all edges $(u \text{ — } v)$ between objects of U and V where $u \in U$, and $v \in V$, the direction of all edges must be either $u \to v$ or $u \leftarrow v$ and not both*

4. *For all edges $u \to v$ between the objects of U and V, out degree of $u = 1$ but indegree of v can be greater than 1.*

In this regard, relational skeleton generation process can be considered as a problem of generating objects and assigning links (or foreign keys) between them such that the resulting graph is a k-partite graph of definition 6. In our previous work [1, 2], we presented an algorithm to generate relational skeleton, where it generates nearly same number of objects of each class and iteratively adds random edges between objects of a pair of classes such that the direction of the edges conform to the underlying schema. This approach, in fact, does not create realistic skeleton because in real world, relational skeleton tends to be scale-free, i.e., degree of the vertices of the graph follows power law. Hence, in real datasets, the number of objects for classes with foreign keys tend to be very high compared to that for classes which do not have foreign keys and are referenced by other classes. Thus, we took a different approach to generate relational skeleton. Our new and improved approach to generating such k-partite graph is presented as Algorithm 5. We adapt [5]'s directed scale-free graph generation algorithm for our special k-partite graph and use Chinese Restaurant Process[29] to apply preferential attachment.

The basic idea here is to iteratively generate an object of a class with no parents in the relational schema DAG and then recursively add an edge from this object to objects of its children classes. This process is essentially a depth first search (DFS), where we begin by generating an object of the root node of the graph and then at each encounter of a node in DFS, we add an edge from the object of the parent node to either a new or an existing object of the encountered node. The object of the parent node gets connected to a new object with probability $p = \alpha/(n_p - 1 + \alpha)$, where n_p is the total of objects of the parent node generated so far, and α is a scalar parameter for the process. When it gets attached to an existing object, an object of the correct type is picked from the set of existing objects with probability $n_k/(n_p - 1 + \alpha)$, where n_k is the indegree of the object to be selected and n is the total number of objects generated so far. Thus, as the skeleton graph grows, probability of getting connected to new objects will decrease and the objects with higher indegree will be preferred for adding new edges. At each iteration, a DFS is performed starting from one of the nodes without parents in the relational schema DAG. Thus, if there is only

Algorithm 5: Generate_Relational_Skeleton

Input: Relational Schema as a DAG, $\mathcal{G} = (V_g, E_g)$; Total number of objects in the resulting skeleton, N_{total}; Scalar parameter for CRP, α

Output: A relational skeleton, $\mathcal{I} = (V, E)$

begin

  for $node \in V_g$ **do**

    $N(node) \leftarrow 0$ *%Total number of objects of each type generated so far*

  $V \leftarrow \{\}$ *%Set of objects*

  $E \leftarrow \{\}$ *%Set of directed edges between objects*

  $m \leftarrow$ Number of nodes without any parents in $\mathcal{G}$ *%number of roots*

  if $m > 1$ **then**

    Divide $\mathcal{G}$ into m subgraphs such that each subgraph contains a root and all of its descendants.

  repeat

    if $m > 1$ **then**

      $g \leftarrow$ one of the m subgraphs picked randomly

    else

      $g \leftarrow \mathcal{G}$ *%i.e., if $\mathcal{G}$ has only one root*

    $obj_{root} \leftarrow$ A new object of the root of g

    $n_{root} \leftarrow n_{root} + 1$

    $children \leftarrow$ Children of the root in g

    $((V', E'), N') \leftarrow$ Generate_SubSkeleton$(obj_{root}, g, children, N, \alpha)$ *%Perform depth-first search over g and add edges recursively*

    $V \leftarrow V \cup V'$

    $E \leftarrow E \cup E'$

    $N \leftarrow N'$ *%Update the set of number of generated objects of each type*

    $n \leftarrow cardinality(V)$ *%Total number of objects generated so far.*

  until $n >= N_{total}$;

  $\mathcal{I} \leftarrow (V, E)$

one node that does not have any parent, then each iteration will visit all classes in the relational schema resulting in a complete set of objects and relations for all classes, otherwise only a subset of classes will be visited

Algorithm 6: Generate_SubSkeleton

Input: Parent object obj_p; Graph g; Parent node, *parent*; Children nodes, *children*; Set of the number of objects of each class generated so far, N; Scalar parameter α

Output: Relational skeleton, $\mathcal{I} = (V, E)$; Set of the number of objects of each class generated so far, N

begin

- $V \leftarrow \{obj_p\}$
- $E \leftarrow \{\}$
- $n_p \leftarrow N(parent)$ *%Total number of parents generated so far*
- **for** $C \in children$ **do**
  - $n_c \leftarrow N(C)$ *%Total number of the child C generated so far*
  - $p \leftarrow \alpha/(n_p - 1 + \alpha)$
  - $r \leftarrow$ A random value between 0 and 1 $\%r \in [0, 1]$
  - **if** $r <= p$ **then**
    - $obj_c \leftarrow$ Create a new object of type C
    - $N(C) \leftarrow n_c + 1$
    - $e_{pc} \leftarrow (obj_p, obj_c)$ *%Add an edge from obj_p to obj_c*
    - $E \leftarrow E \cup \{e_{pc}\}$
    - $children_c \leftarrow$ Children of C in the graph g
    - $((V', E'), N') \leftarrow$ Generate_SubSkeleton$(obj_c, g, C, children_c, N, \alpha)$
    - $V \leftarrow V \cup V'$
    - $E \leftarrow E \cup E'$
    - $N \leftarrow N'$
  - **else**
    - $obj_c \leftarrow$ An existing object of type C picked randomly with probability $n_k/(n_p - 1)$ where $n_k =$ indegree of obj_c
    - $e_{pc} \leftarrow (obj_p, obj_c)$ *%Add an edge from obj_p to obj_c*
    - $E \leftarrow E \cup \{e_{pc}\}$
- $\mathcal{I} \leftarrow (V, E)$

in each iteration. So, at the beginning of each iteration, one of the nodes without parents is picked randomly in the latter case. The iteration process is continued until the skeleton contains the required number of objects.

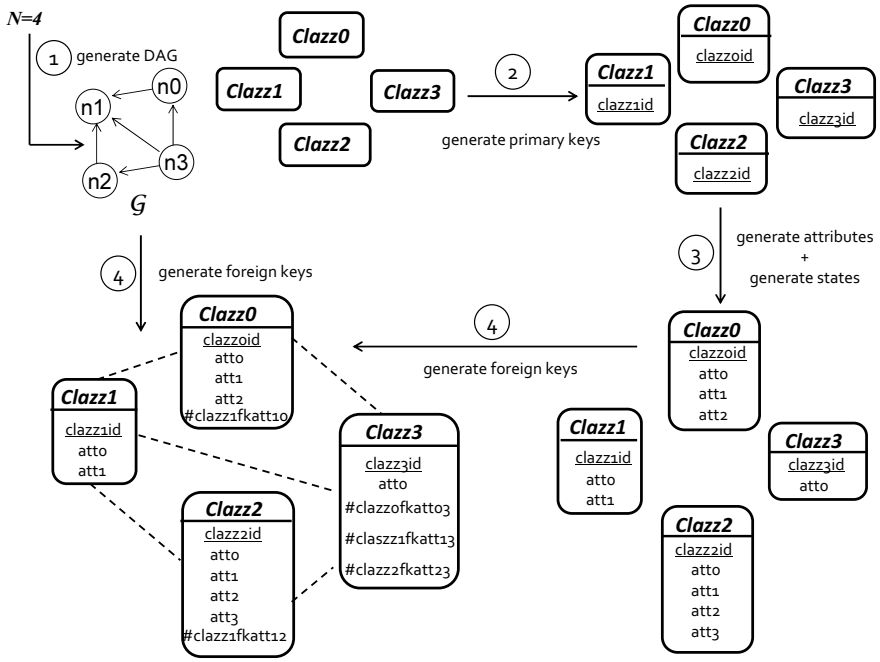


Figure 5: Relational schema generation steps

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 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 a 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 attributes $att0, att1$ 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 $clazz1fkatt12$).

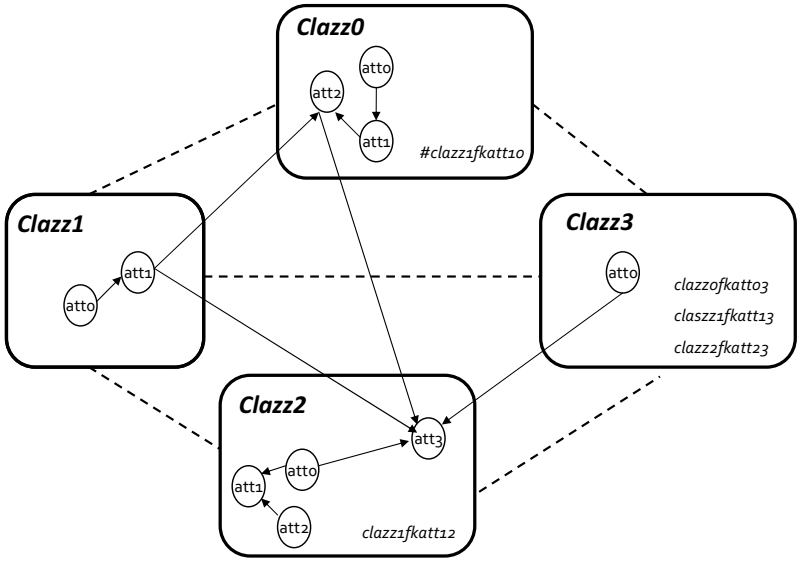


Figure 6: Graph dependency structure generation

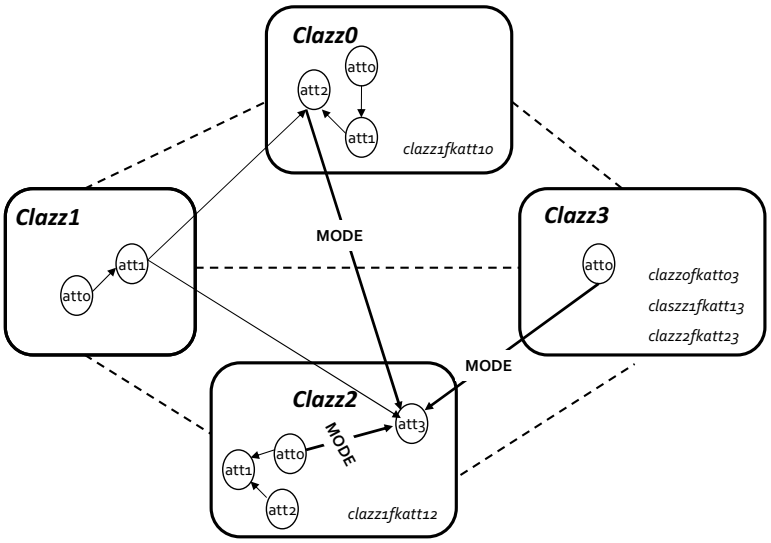


Figure 7: Example of a generated relational schema where the dotted lines represent referential constraints and the generated PRM dependency structure where the arrows represent probabilistic dependencies. Here, we have excluded slot chains to not overload the figure. Details about slot chains from which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

PRM generation. We recall that this process is performed in 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 determine 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.
Three of the inter-class dependencies have been generated from slot chains of length 1:
$Clazz0.clazz1fkatt10.att1 \rightarrow Clazz0.att2$;
$MODE(Clazz2.clazz2fkatt23^{-1}.att0) \rightarrow Clazz2.att3$ and;
$Clazz2.clazz1fkatt12.att1 \rightarrow Clazz2.att3$
One from slot chain of length 2:
$MODE(Clazz2.clazz1fkatt12.clazz1fkatt12^{-1}.att0) \rightarrow Clazz2.att3$
One from slot chain of length 3:
$MODE(Clazz2.clazz2fkatt23^{-1}.claszz1fkatt13.clazz1fkatt10^{-1}.att2) \rightarrow Clazz2.att3$

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 create a relational skeleton for the relational schema by performing depth first search on the schema DAG (cf. Algorithm 5). The first three iterations of the DFS are shown in figures 8 and 9. As the schema has only one node without any parent (i.e., a class without any foreign key), one complete DFS returns a set of objects of each class as shown in figure 8. At each iteration, we obtain different number of objects. As we can see in figure 9, the first iteration created five objects whereas the second and third iteration resulted in four and two objects respectively. We continue the iteration until we obtain the required number of objects in the skeleton. We then instantiate the probabilistic model generated in the previous step with the generation skeleton to obtain a ground Bayesian network. Sampling this GBN enables us to populate values for all attributes of all objects in the relational skeleton. For this example, we generated a random dataset with 2500 objects. The corresponding schema diagram is shown in figure 10, which also shows the number of objects of each class. The diagram is generated using SchemaSpy[9].

[9] http://schemaspy.sourceforge.net/

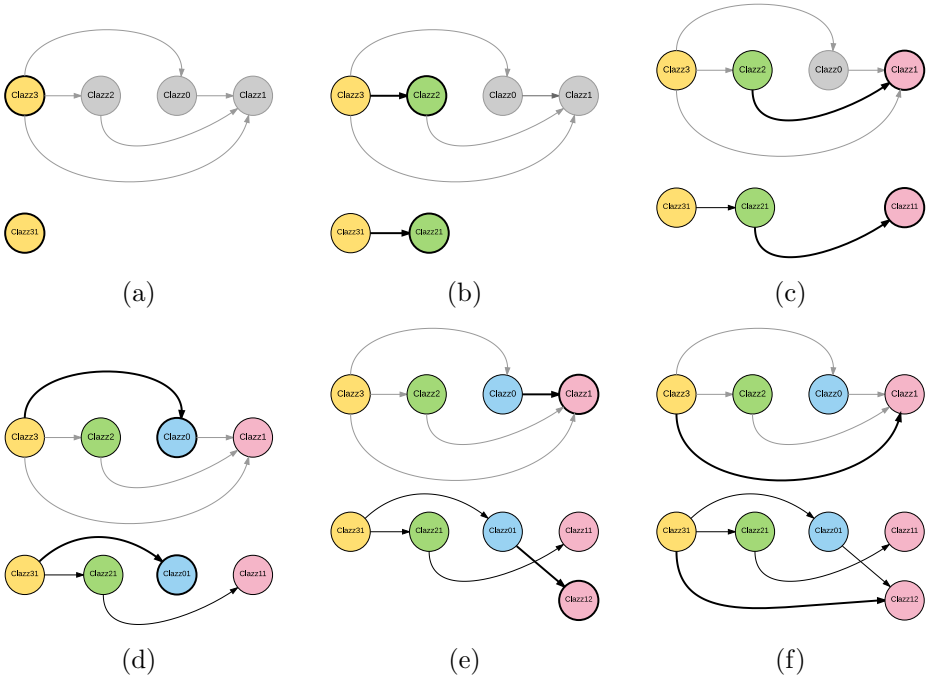
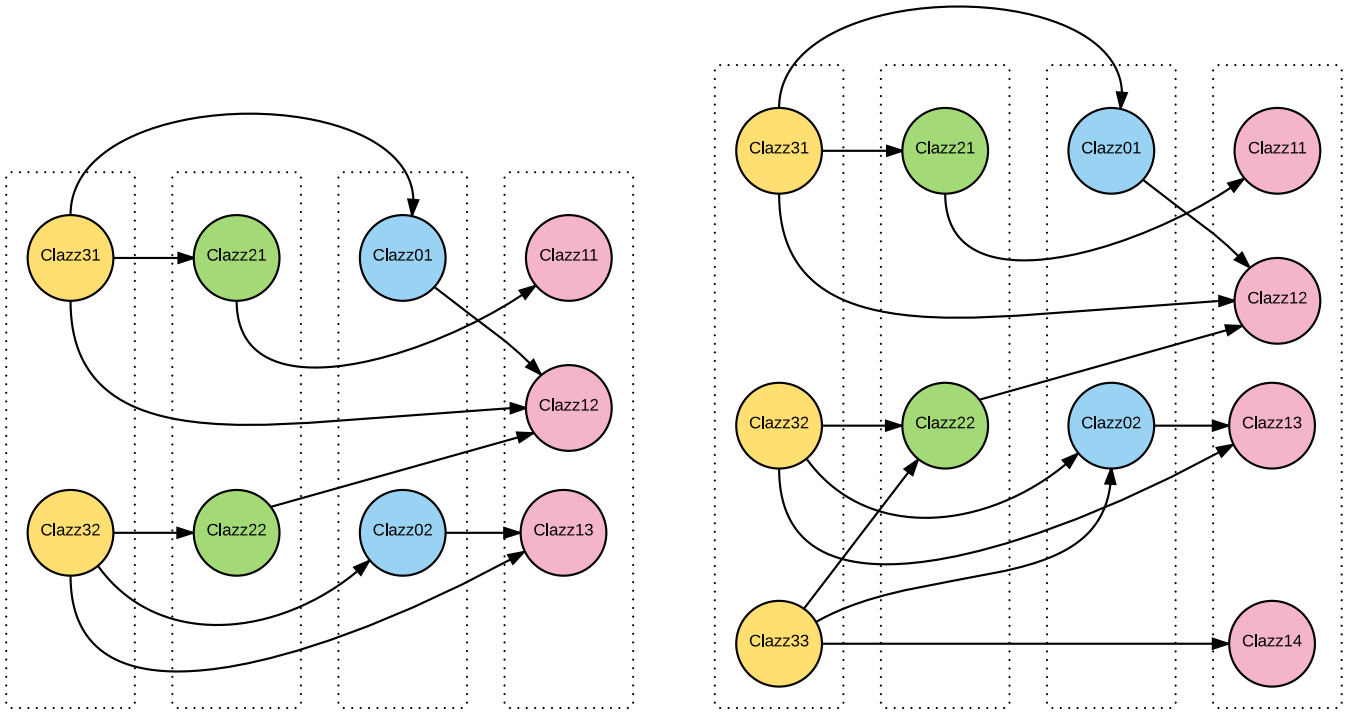


Figure 8: Generating objects while performing Depth First Search (DFS) on the relational schema. This figure shows one iteration of a DFS performed on the schema of figure 5. In each of the sub-figures above, the upper graph is the concerned relational schema and the lower graph is the relational skeleton being generated. The node (and the edge) encountered at each step of the DFS is shown by thick lines. Colors are used to only as a visual aid to distinguish between objects of different classes and add no significant meaning to the process. (8a) We begin by creating a new object of the node 'Class3' as it does not have any parent. (8b) Then, we traverse to one of the children of this node in the schema ('Class2' here). As there is no object of this class so far, a new object will be created. (8c) and (8d) Now, continuing the DFS, we encounter the node 'Class1' (the child of 'Class2'), and then 'Class0' (a child of 'Class3'). Like earlier, new objects of 'Class1' and 'Class0' will be generated. (8e) As 'Class0' has a child, we reach 'Class1'. At this step, an object of 'Class1' is already present. So the object 'Class01' can either create a new object or get connected to 'Class11'. In this example, it gets linked to a new object 'Class12'. (8f) In the next step of the DFS, 'Class1' is encountered again as it is a child of 'Class3'. Here, 'Class31' gets attached to an existing object of 'Class1'.



(a) (b)

Figure 9: Next two iterations of DFS on the relational schema of figure 5 following the first iteration of figure 8 to generate relational skeleton graph. At each iteration, a new object of 'Class3' will always be generated as it does not have any parent. The object will then be linked to an existing object or a new one, and the same thing goes on for the new objects. Here, the skeleton after the first iteration has five objects. The second iteration creates four new objects, whereas the third iteration creates only two objects.

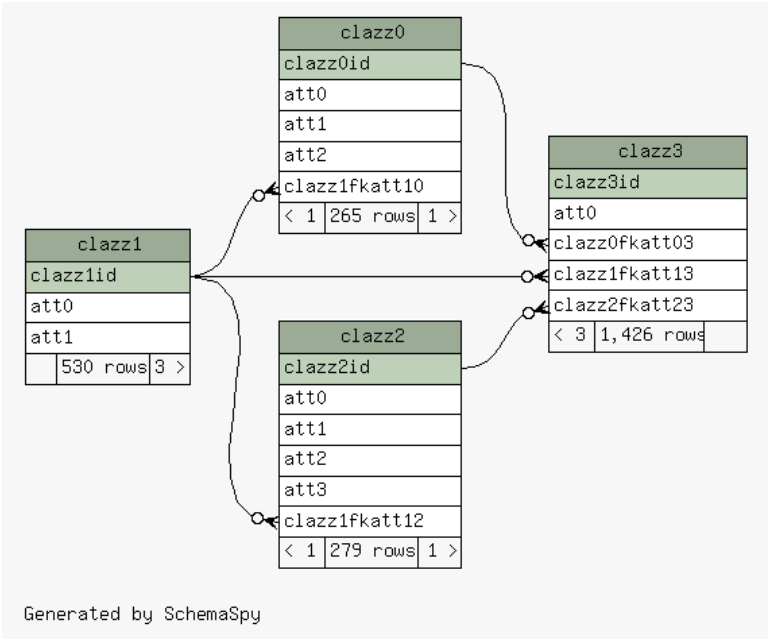


Figure 10: Schema diagram of our toy example showing the number of objects/rows in each class.

5 Implementation

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

5.1 Software implementation

The proposed algorithms have been implemented in PILGRIM[10] API, a software platform that our lab is actively developing to provide an efficient tool to deal with several probabilistic graphical models (e.g., BNs, Dynamic BNs, PRMs). Developed in C++, PILGRIM uses Boost graph library[11] to manage graphs, ProBT API[12] to manipulate BNs objects and Database Template Library (DTL)[13] to communicate with databases. Currently, only PostgreSQL RDBMS is supported in this platform.

Besides the algorithms, we have also implemented serialization of PRMs. Because there is currently no formalization of PRMs, we propose an enhanced version of the XML syntax of the *ProbModelXML* specification[14] to serialize our generated models. We have added new tags to specify

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

notions related to relational schema definition and we used the standard `<AdditionalProperties>` tags to add further notions related to PRMs (e.g., aggregators associated with dependencies, classes associated with nodes).

5.2 Implemented policies

Policy for generating the relational schema DAG structure. To randomly generate the relational schema DAG structure, we use PMMixed algorithm (cf. Section 2.1), which generates uniformly distributed DAGs in the DAGs space. The structure generated by this algorithm may be a disconnected graph whereas 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 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 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.

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 explores 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. The relational skeleton generation algorithm is basically an iteration of depth first search over a relational schema. Thus, complexity of the algorithm would be the same as that of a DFS, i.e. $\mathcal{O}(V + E)$ where V and E are respectively the number of vertices and the number of edges in the graph.

6 Conclusion and perspectives

We have developed a process that allows 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 the relational schema but also from the probabilistic dependencies present in the PRM.

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 test query components to help database designers to evaluate the effectiveness of their RDBMS components.

References

[1] 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.

[2] M. Ben Ishak, P. Leray, and N. Ben Amor. Probabilistic relational model benchmark generation: Principle and application. *Intelligent Data Analysis International Journal (to appear)*, pages ?–?, 2016.

[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] B. Bollobás, C. Borgs, J. Chayes, and O. Riordan. Directed scale-free graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 132–139. Society for Industrial and Applied Mathematics, 2003.

[6] 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.

[7] 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.

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

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

[12] 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.

[13] 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.

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

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

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

[23] 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.

[24] 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.

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

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

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

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

[29] J. Pitman. Combinatorial stochastic processes. *Lecture Notes for St. Flour Summer School*, 2002.

[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.